TUGAS AKHIR - KI141502

MODIFIKASI *ROUTE DISCOVERY* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS *ONE-HOP NODE* DI VANETS

AVIANANDA DWIRAHMA JULIARTI
NRP 5114100085

Dosen Pembimbing I
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

Dosen Pembimbing II
Prof. Ir. Supeno Djanali, M.Sc, Ph.D.

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR – KI141502

MODIFIKASI *ROUTE DISCOVERY* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS *ONE-HOP NODE* DI VANETS

AVIANANDA DWIRAHMA JULIARTI
NRP 5114100085

Dosen Pembimbing I
Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.

Dosen Pembimbing II
Prof. Ir. Supeno Djanali, M.Sc, Ph.D.

Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2017

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESES – KI141502

MODIFICATION OF ROUTE DISCOVERY PROCESS IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) BASED ON ONE-HOP NODE CONNECTIVITY LEVEL IN VANETS

AVIANANDA DWIRAHMA JULIARTI
NRP 5114100085

First Advisor
Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.

Second Advisor
Prof. Ir. Supeno Djanali, M.Sc, Ph.D.

Department of Informatics
Faculty of Information Technology and Communication
Sepuluh Nopember Institute of Technology
Surabaya 2017

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

MODIFIKASI *ROUTE DISCOVERY* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS *ONE-HOP NODE* DI VANETS

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
AVIANANDA DWIRAHMA JULIARTI
NRP: 5114100085

Disetujui oleh Pembimbing Tugas Akhir:

1. Dr.Eng. Radityo Anggoro, S.Kom., M.Sc. .....
(NIP. 198410162008121002) (Pembimbing 1)

2. Prof. Ir. Supeno Djanali, M.Sc., Ph.D. .....
(NIP. 194806191973011001) (Pembimbing 2)

SURABAYA
JANUARI, 2018

v

(Halaman ini sengaja dikosongkan)

MODIFIKASI *ROUTE DISCOVERY* PADA *AD-HOC ON DEMAND DISTANCE VECTOR* (AODV) BERDASARKAN LEVEL KONEKTIVITAS *ONE-HOP NODE* DI VANETS

Nama Mahasiswa	**: Aviananda Dwirahma Juliarti**
NRP	**: 5114100085**
Departemen	**: Informatika FTIK-ITS**
Dosen Pembimbing 1	**: Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Dosen Pembimbing 2	**: Prof. Ir. Supeno Djanali, M.Sc, Ph.D.**

Abstrak

Vehicular Ad hoc Networks (VANETs) merupakan pengembangan dari *Mobile Ad hoc Network* (MANET) dimana *node* memiliki karakteristik dengan mobilitas yang sangat tinggi dan terbatas pada pola pergerakannya. Ada banyak *routing protocol* yang dapat diimplementasikan pada VANETs, salah satunya adalah *Ad hoc On demand Distance Vector* (AODV).

AODV merupakan salah satu *routing protocol* yang termasuk dalam klasifikasi *reactive routing protocol*, sebuah protokol yang hanya akan membuat rute ketika *node* sumber membutuhkannya. AODV memiliki dua fase, yaitu *route discovery* dan *route maintenance. Route discovery* digunakan untuk meminta dan meneruskan informasi rute yang terdiri dari proses pengiriman *Route Request* (RREQ) dan *Route Reply* (RREP), sedangkan *route maintenance* digunakan untuk mengetahui informasi adanya kesalahan pada rute. Pada fase ini terdapat proses pengiriman *Route Error* (RERR).

Modifikasi pada Tugas Akhir ini akan dilakukan pada proses *route discovery* berdasarkan level konektivitas *one-hop node*, yaitu dengan cara mengeliminasi jumlah *forwarding node* yang bertugas untuk mengirim ulang *(re-broadcast)* RREQ. Hal ini dilakukan agar dapat meningkatkan kinerja protokol AODV untuk mencari rute yang stabil dengan cara memodifikasi beberapa bagian

dari mekanisme pengiriman paket RREQ. Dari hasil uji coba, AODV yang dimodifikasi berhasil meningkatkan nilai *Packet Delivery Ratio* (PDR) hingga 9,19%, penurunan nilai *Routing Overhead* (RO) hingga 2,17%, dan penurunan nilai *Forwarded Route Request* (RREQ F) hingga 31,62%.

Kata kunci: VANETs, AODV, NS2, SUMO, One-Hop Node, Forwarding Node, Node Tetangga

MODIFICATION OF ROUTE DISCOVERY IN AD-HOC ON DEMAND DISTANCE VECTOR (AODV) BASED ON ONE-HOP NODE CONNECTIVITY LEVEL IN VANETS

Student's Name	**:**	**Aviananda Dwirahma Juliarti**
Student's ID	**:**	**5114100085**
Department	**:**	**Informatics – FTIK ITS**
First Advisor	**:**	**Dr.Eng. Radityo Anggoro, S.Kom., M.Sc.**
Second Advisor	**:**	**Prof. Ir. Supeno Djanali, M.Sc., Ph.D.**

Abstract

VANETs are an improvement of MANET which have high mobility node characteristic and limited movement pattern. There are many routing protocols that can be implemented on VANETs and one of them is AODV.

AODV is an example of reactive routing protocol classification, a protocol that will only create a route when the source node needs it. AODV have 2 phase which are route discovery and route maintenance. Route discovery is used for requesting and forwarding a route information that consist of Route Request (RREQ) and Route Reply (RREP), meanwhile route maintenance that consist of Route Error (RERR) is used for finding out an error information in route.

In this final project, a modification of AODV routing protocol has been proposed in route discovery based on one-hop node connectivity level by eliminating number of one-hop node that eligible rebroadcast a RREQ. This is done to improve the performance of the AODV routing protocol to find a stable route by modifying some parts of the RREQ packet delivery mechanism. The evaluation shows that, the value of Packet Delivery Ratio (PDR) has increased by 9,19%, the value of Routing Overhead (RO) has

decreased by 2,17%, and the value of Forwarded Route Request (RREQ F) has decreased by 31,62%.

Keyword: VANETs, AODV, NS2, SUMO, One-Hop Node, Forwarding Node, Neighbor Node

KATA PENGANTAR

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya penulis dapat menyelesaikan Tugas Akhir yang berjudul **"Modifikasi *Route Discovery* pada *Ad-hoc On Demand Distance Vector* (AODV) berdasarkan Level Konektivitas *One-Hop Node* di VANETs".**

Harapan penulis semoga apa yang tertulis di dalam buku Tugas Akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT atas semua rahmat yang diberikan sehingga penulis dapat menyelesaikan Tugas Akhir ini.
2. Bapak Ary Pibyanto dan Ibu Erna Damayanti selaku kedua orangtua penulis atas segala dukungan berupa motivasi serta doa sehingga penulis dapat mengerjakan Tugas Akhir ini.
3. Shabrina Dessy A. dan M. Farhan Raffyanto selaku kakak dan adik penulis atas segala dukungan yang telah diberikan sehingga penulis tetap semangat dalam mengerjakan Tugas Akhir ini.
4. Bapak Dr. Eng. Radityo Anggoro, S.Kom., M.Sc., dan Bapak Prof. Ir. Supeno Djanali, M.Sc, Ph.D. selaku dosen pembimbing pertama, atas arahan dan bantuannya dalam pengerjaan Tugas Akhir ini.
5. Teman seperjuangan semasa kuliah (Zahrah Citra, Desy Nurbaiti, Raras Anggita, Nafia Rizky Y., Rahmatin Nadia, Tiara Anggita, Kania Amalia, Vinsensia Sipriana, Anindita Larasati, Anne Annisa) dan teman – teman angkatan 2014 yang selama ini sudah membantu penulis dalam menyelesaikan Tugas Akhir ini
6. Keluarga AJK 2013 – 2016 (Mas Wicak, Mas Daniel. Mas Setiyo, Mas Uul, Mba Nindy, Mba Risma, Mba Zaza,

Syukron, Oing, Fatih, Thoni, Afif, Bebet, Awan, Didin, Fuad, Nahda, Daus, Hana, Penyok, Satria, Khawari, Raldo, dan Router) yang sudah menyemangati penulis dalam pengerjaan Tugas Akhir ini.

7. Sahabat Penulis semasa kuliah (Aldo, Nezar, Faris, Glleen, Fikry, Hilman, Rage, Haidar, Sani, Lian, Dito, Dio, Hari, Rian, Pras, Buyung, Paul, Faiq, Tras, Hamka, Cimeng, Pentol, dan Upil) yang selama ini sudah menghibur Penulis semasa kuliah di ITS.
8. Teman – teman ITS SMANITRA yang selama ini sudah menyemangati Penulis dalam menjalankan kuliah di ITS.
9. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis menyadari bahwa masih terdapat kekurangan, kesalahan, maupun kelalaian yang telah penulis lakukan. Oleh karena itu, saran dan kritik yang membangun sangat dibutuhkan untuk penyempurnaan Tugas Akhir ini.

Surabaya, Januari 2018

Aviananda Dwirahma

DAFTAR ISI

Abstrak ..... **vii**
Abstract ..... **ix**
KATA PENGANTAR ..... **xi**
DAFTAR ISI ..... **xiii**
DAFTAR GAMBAR ..... **xvii**
DAFTAR TABEL ..... **xix**
BAB I PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 3
1.3 Batasan Permasalahan 3
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Metodologi 4
1.6.1 Penyusunan Proposal Tugas Akhir 4
1.6.2 Studi Literatur 4
1.6.3 Analisis dan Desain Sistem 4
1.6.4 Implementasi Sistem 5
1.6.5 Pengujian dan Evaluasi 5
1.6.6 Penyusunan Buku 5
1.7 Sistematika Penulisan Laporan 5
BAB II TINJAUAN PUSTAKA **7**
2.1 VANETs 7
2.2 *Ad-hoc On demand Distance Vector* (AODV) 8
2.3 *Network Simulator-2* (NS-2) 10
2.3.1 Instalasi 11
2.3.2 *Trace File* 11
2.4 OpenStreetMap 13
2.5 *Java OpenStreetMap Editor* (JOSM) 14
2.6 *Simulation of Urban Mobility* (SUMO) 14
2.7 AWK 16
BAB III PERANCANGAN **17**
3.1 Deskripsi Umum 17
3.2 Perancangan Skenario Mobilitas 20

3.2.1 Perancangan Skenario *Grid* 20
3.2.2 Perancangan Skenario *Real* 22
3.3 Perancangan Modifikasi *Routing Protocol* AODV 23
3.3.1 Perancangan Penghitungan Jumlah *Node* Tetangga untuk Setiap *Node* 24
3.3.2 Perancangan Pemilihan *Forwarding Node* 25
3.4 Perancangan Simulasi pada NS-2 26
3.5 Perancangan Metrik Analisis 26
3.5.1 *Packet Delivery Ratio* (PDR) 27
3.5.2 *Average End-to-End Delay* (E2E) 27
3.5.3 *Routing Overhead* (RO) 28
3.5.4 *Forwarded Route Request* (RREQ F) 28
BAB IV IMPLEMENTASI 29
4.1 Implementasi Skenario Mobilitas 29
4.1.1 Skenario *Grid* 29
4.1.2 Skenario *Real* 33
4.2 Implementasi Modifikasi pada *Routing Protocol* AODV untuk Menentukan *Forwarding Node* 35
4.2.1 Implementasi Penghitungan Jumlah *Node* Tetangga 35
4.2.2 Implementasi Pemilihan *Forwarding Node* 38
4.3 Implementasi Simulasi pada NS-2 39
4.4 Implementasi Metrik Analisis 41
4.4.1 Implementasi *Packet Delivery Ratio* (PDR) 41
4.4.2 Implementasi *Average End-to-End Delay (E2E)* 43
4.4.3 Implementasi *Routing Overhead* (RO) 44
4.4.4 Implementasi *Forwarded Route Request* (RREQ F) 44
BAB V UJI COBA DAN EVALUASI 47
5.1 Lingkungan Uji Coba 47
5.2 Hasil Uji Coba 48
5.2.1 Hasil Pra Uji Coba Penentuan *Threshold* 48
5.2.2 Hasil Uji Coba Skenario *Grid* 50
5.2.3 Hasil Uji Coba Skenario *Real* 57
BAB VI KESIMPULAN DAN SARAN 65
6.1 Kesimpulan 65
6.2 Saran 65

DAFTAR PUSTAKA **67**
LAMPIRAN **69**
A.1 Kode Fungsi addNeighbor() 69
A.2 Kode Fungsi recvHello() 70
A.3 Kode Fungsi recvRequest() 71
A.4 Kode Skenario NS-2 76
A.5 Kode Konfigurasi *Traffic* 79
A.6 Kode Skrip AWK *Packet Delivery Ratio* 80
A.7 Kode Skrip AWK Rata-Rata *End-to-End Delay* 81
A.8 Kode Skrip AWK *Routing Overhead* 82
A.9 Kode Skrip AWK *Forwarded Route Request* 83
BIODATA PENULIS **85**

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Gambar 2.1 Ilustrasi VANETs [13] 8
Gambar 2.2 Ilustrasi pencarian rute routing protocol AODV [14] 9
Gambar 2.3 Perintah untuk menginstall dependency NS-2 11
Gambar 2.4 Baris kode yang diubah pada file ls.h 11
Gambar 3.1 Diagram Rancangan Simulasi AODV Asli 17
Gambar 3.2 Diagram Rancangan Simulasi AODV Modifikasi ... 18
Gambar 3.3 Alur Pembuatan Skenario Grid 21
Gambar 3.4 Alur Pembuatan Skenario Real 23
Gambar 3.5 Pseudocode Penghitungan Jumlah Node Tetangga .. 25
Gambar 3.6 Pseudocode Pemilihan Forwarding Node 26
Gambar 4.1 Perintah netgenerate 29
Gambar 4.2 Hasil Generate Peta Grid 30
Gambar 4.3 Perintah randomTrips 31
Gambar 4.4 Perintah duarouter 31
Gambar 4.5 File Skrip .sumocfg 32
Gambar 4.6 Perintah SUMO untuk membuat skenario .xml 32
Gambar 4.7 Perintah traceExporter 33
Gambar 4.8 Ekspor Peta dari OpenStreetMap 33
Gambar 4.9 Perintah netconvert 34
Gambar 4.10 Hasil Konversi Peta Real 34
Gambar 4.11 Potongan Kode Memasukkan Node Tetangga ke dalam List 36
Gambar 4.12 Potongan Kode Memanggil fungsi addNeighbor ()37
Gambar 4.13 Potongan Kode Menghitung Jumlah Node Tetangga 38
Gambar 4.14 Potongan Kode Penyeleksian Forwarding Node 39
Gambar 4.15 Implementasi Simulasi NS-2 40
Gambar 4.16 Implementasi Simulasi File Traffic 41
Gambar 4.17 Pseudocode untuk Menghitung PDR 42
Gambar 4.18 Pseudocode untuk Perhitungan Rata-Rata E2E 43
Gambar 4.19 Pseudocode untuk Perhitungan Routing Overhead 44
Gambar 4.20 Pseudocode Perhitungan Forwarded Route Request 45
Gambar 5.1 Grafik PDR Skenario Grid 52

Gambar 5.2 Grafik E2E Skenario Grid.....53
Gambar 5.3 Grafik Routing Overhead Skenario Grid 54
Gambar 5.4 Grafik Forwarded Route Request Skenario Grid.....56
Gambar 5.5 Grafik Rata - Rata PDR Skenario Real.....59
Gambar 5.6 Grafik E2E pada Skenario Real 60
Gambar 5.7 Grafik Rata - Rata RO Skenario Real.....61
Gambar 5.8 Grafik Rata - Rata RREQ F Skenario Real.....62

DAFTAR TABEL

Tabel 2.1 Detail Penjelasan Trace File AODV.....12
Tabel 3.1 Daftar Istilah.....19
Tabel 5.1 Spesifikasi Perangkat yang Digunakan 47
Tabel 5.2 Lingkungan Uji Coba 48
Tabel 5.3 Hasil Pra Uji Coba di Lingkungan Jarang 49
Tabel 5.4 Hasil Pra Uji Coba di Lingkungan Sedang.....49
Tabel 5.5 Hasil Pra Uji Coba di Lingkungan Padat.....49
Tabel 5.6 Hasil Rata - Rata PDR Skenario Grid.....51
Tabel 5.7 Hasil Rata - Rata E2E Skenario Grid 51
Tabel 5.8 Hasil Rata - Rata RO Skenario Grid.....51
Tabel 5.9 Hasil Rata - Rata RREQ F Skenario Grid 51
Tabel 5.10 Hasil Rata - Rata Perhitungan PDR pada Skenario Real 58
Tabel 5.11 Hasil Rata -Rata Perhitungan E2E pada Skenario Real 58
Tabel 5.12 Hasil Rata - Rata Perhitungan RO pada Skenario Real 58
Tabel 5.13 Hasil Rata - Rata Perhitungan RREQ F pada Skenario Real.....58

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Saat ini perkembangan teknologi informasi dan komunikasi menjadi salah satu indikator kemajuan peradaban manusia. Salah satu teknologi yang membantu manusia dalam berkomunikasi dengan mudah adalah *Vehicular Ad hoc Networks* (VANETs). VANETs merupakan suatu mekanisme yang dapat menghubungkan kendaraan satu dengan yang lainnya menggunakan jaringan nirkabel. VANETs dapat berguna pada banyak hal, seperti mengemudi secara otomatis, navigasi, pencegahan kecelakaan yang dapat meningkatkan keamanan berkendara, serta dapat mengurangi kemacetan lalu lintas [1].

VANETs merupakan pengembangan dari *Mobile Ad hoc Network* (MANET) dimana *node* memiliki karakteristik mobilitas yang sangat tinggi dan terbatas pada pola pergerakannya. Hal tersebut membuat topologi jaringan VANETs lebih dinamis dibandingkan dengan MANET. Mobilitas *node* yang sangat tinggi pada VANETs berpengaruh besar terhadap perubahan topologi jaringan setiap waktu. Selain menyebabkan perubahan topologi, hal tersebut juga dapat menyebabkan rute terputus karena *node* keluar dari jangkauan sinyal transmisi [2].

Ada dua model *routing protocol* yang dapat diterapkan dalam VANETs, yaitu *proactive* dan *reactive routing*. *Proactive routing* adalah protokol yang bekerja dengan cara membentuk tabel *routing* dan melakukan *update* setiap saat pada selang waktu tertentu contohnya adalah *Optimized Link State Routing Protocol* (OLSR). Sedangkan *reactive routing*

merupakan mekanisme *routing* yang membentuk tabel *routing* jika ada permintaan pengiriman data atau terjadinya perubahan rute dalam setiap jaringan. Contoh *reactive routing protocol* adalah *Ad hoc On-Demand Distance Vector* (AODV), *Dynamic Source Routing* (DSR), dan *Temporary Order Routing Algortihm* (TORA).

Pencarian rute menjadi suatu mekanisme yang penting untuk mendukung mobilitas di VANETs. Pemilihan rute yang tepat saat proses pencarian rute sangat diperlukan untuk memperpanjang waktu penggunaan rute. Salah satu cara dapat dilakukan adalah dengan memilih rute yang memiliki potensi kecil terputus [3].

Pada Tugas Akhir ini diusulkan suatu mekanisme *routing discovery* pada *reactive routing* AODV untuk memperoleh rute berdasarkan level konektivitas *one-hop* pada VANETs. Penulis mengadaptasi cara kerja OLSR dimana hanya beberapa *node* yang terpilih sebagai *node* perantara untuk meneruskan paket dan mengurangi *control packet* yang terlalu banyak [4]. *Nod*e tersebut hanya memilih beberapa *node* tetangga yang akan menjadi *node* perantara, dan hanya *node* perantara yang dapat meneruskan *Route Request* (RREQ), sedangkan *node* lainnya tidak akan meneruskan (*drop*) paket tersebut. Metode yang digunakan untuk mendeteksi node tetangga menggunakan HELLO *messages* yang terdapat pada AODV yang dikirim secara periodic untuk mengetahui simpul tetangga pada suatu waktu [5]. Hasil akhir yang diharapkan adalah mengetahui perbandingan kinerja antara AODV dan AODV yang telah dimodifikasi diukur berdasarkan performansi *Packet Delivery Ratio* (PDR), *Routing Overhead*, *End-to-End Delay*, dan *Forwarded Route Request* (RREQ F).

1.2 Rumusan Masalah

Tugas Akhir ini mengangkat beberapa rumusan masalah sebagai berikut:

1. Bagaimana membatasi jumlah *forwarding node* dalam proses *rebroadcast Route Request* (RREQ)?
2. Bagaimana dampak batasan *forwarding node* terhadap performa protokol AODV secara keseluruhan?

1.3 Batasan Permasalahan

Permasalahan yang dibahas pada Tugas Akhir ini memiliki batasan sebagai berikut:

1. Jaringan yang digunakan adalah jaringan *Vehicular Ad hoc Networks* (VANETs).
2. *Routing protocol* yang diujicobakan yaitu AODV.
3. Simulasi pengujian jaringan menggunakan *Network Simulator 2* (NS-2).
4. Pembuatan skenario uji coba menggunakan *Simulation of Urban Mobility* (SUMO).

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mereduksi jumlah *forwarding node* yang bertanggung jawab untuk *rebroadcast* RREQ *message*.
2. Mengurangi jumlah *control packet* yang di*broadcast* dalam jaringan.

1.5 Manfaat

Manfaat yang diperoleh dari pengerjaan Tugas Akhir ini adalah dapat memberikan informasi tentang dampak dari pembatasan *forwarding node* berdasarkan level konektivitas *one-*

hop node terhadap kinerja *routing protocol* AODV di lingkungan VANETs.

1.6 Metodologi

Pembuatan Tugas Akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Tahapan awal dari Tugas Akhir ini adalah penyusunan Proposal Tugas Akhir. Proposal Tugas Akhir berisi pendahuluan, deskripsi dan gagasan metode-metode yang dibuat dalam Tugas Akhir ini. Pendahuluan ini terdiri atas hal yang menjadi latar belakang diajukannya Tugas Akhir, rumusan masalah yang diangkat, batasan masalah untuk Tugas Akhir, dan manfaat dari hasil pembuatan Tugas Akhir ini. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan Tugas Akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan Tugas Akhir.

1.6.2 Studi Literatur

Pada tahap ini, dipelajari sejumlah referensi yang diperlukan dalam melakukan implementasi yaitu mengenai VANETs, AODV, NS2, OpenStreetMap, Java OpenStreetMap (JOSM), SUMO, dan AWK.

1.6.3 Analisis dan Desain Sistem

Pada tahap ini dilakukan analisis dari hasil percobaan modifikasi AODV yang dibuat. Data yang dianalis berasal dari perhitungan *Packet Delivery Ratio (*PDR), *Routing Overhead* (RO), *Forwarded Route Request* (RREQ F), dan *End-to-End Delay* paket dari *node* ke *node* lainnya. Hal ini bertujuan untuk merumuskan solusi yang tepat untuk konfigurasi AODV yang

dimodifikasi dalam lingkungan topologi MANET. Setelah selesai diaplikasikan pada MANET, dilakukan simulasi yang dilakukan pada VANETs dengan bantuan SUMO.

1.6.4 Implementasi Sistem

Implementasi merupakan tahap untuk membangun metode-metode yang sudah diajukan pada proposal Tugas Akhir. Pada tahap ini dilakukan implementasi menggunakan NS-2 sebagai *simulator*, Bahasa C/C++ sebagai bahasa pemrograman, dan SUMO sebagai *tools* untuk uji coba dan mengimplentasikan desain sistem yang sudah dirancang.

1.6.5 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian menggunakan SUMO, sebuah *traffic generator* untuk membuat simulasi keadaan topologi yang diujikan. Hasil dari SUMO tersebut akan dijalankan pada NS-2 yang akan menghasilkan *trace file*. *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), *Forwarded Route Request*, dan *End-to-End Delay* akan dihitung dari *trace file* tersebut untuk menguji performa AODV yang telah dimodifikasi.

1.6.6 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku yang menjelaskan seluruh konsep, teori dasar dari metode yang digunakan, implementasi, serta hasil yang telah dikerjakan sebagai dokumentasi dari pelaksanaan Tugas Akhir.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan Tugas Akhir adalah sebagai berikut:

1. Bab I. Pendahuluan

Bab ini berisikan penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan Tugas Akhir.

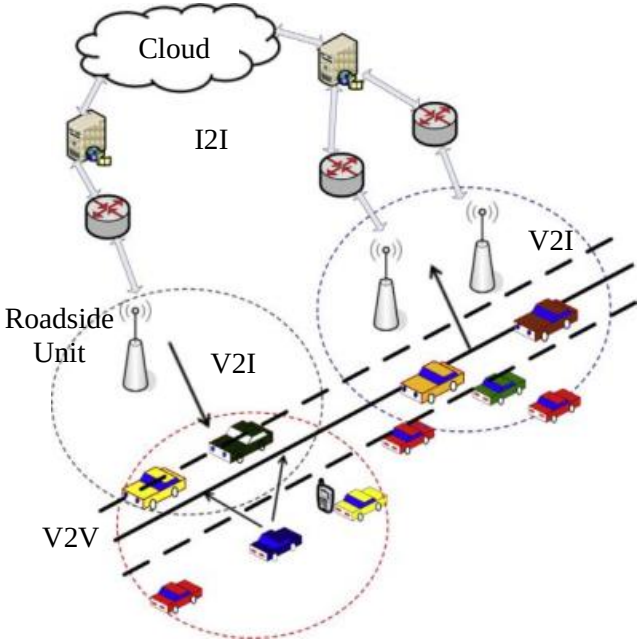
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan Tugas Akhir ini. Bab ini berisi tentang penjelasan singkat mengenai VANETs, AODV, NS2, OpenStreetMap, Java OpenStreetMap (JOSM), SUMO, dan AWK.
3. Bab III. Perancangan
   Bab ini berisi pembahasan mengenai perancangan skenario mobilitas *grid* dan *real*, perancangan simulasi pada NS2, perancangan modifikasi AODV, serta perancangan metrik analisis (*Packet Delivery Ratio, Routing Overhead, Forwarded Route Request,* dan *End-to-End Delay*).
4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk kode sumber dari proses modifikasi protokol AODV, pembuatan simulasi pada NS2, SUMO, dan perhitungan metrik analisis.
5. Bab V. Uji Coba dan Evaluasi
   Bab ini berisikan hasil uji coba dan evaluasi dari implementasi yang telah dilakukan untuk menyelesaikan masalah yang dibahas pada Tugas Akhir.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan Tugas Akhir, dan saran untuk pengembangan solusi ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam Tugas Akhir.
8. Lampiran
   Dalam lampiran terdapat tabel-tabel data hasil uji coba dan kode sumber program secara keseluruhan.

BAB II
TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar atau penjelasan dari metode dan *tools* yang digunakan dalam Tugas Akhir.

2.1 VANETs

Vehicular Ad-hoc Networks (VANETs) merupakan pengembangan dari *Mobile Ad-hoc Network* (MANET) dimana pengembangannya difokuskan pada kendaraan (*vehicle*) yang dapat saling berkomunikasi maupun mengirimkan data. VANETs adalah sebuah teknologi baru yang memadukan kemampuan komunikasi nirkabel kendaraan menjadi sebuah jaringan yang bebas infrastuktur serta memiliki karakteristik mobilitas yang sangat tinggi dan terbatas pada pola pergerakannya. *Node* dalam jaringan dianggap sebagai *router* yang bebas bergerak dan bebas menentukan baik menjadi *client* maupun menjadi *router*. Protokol *routing* pada VANETs memiliki dua model yaitu protokol *reactive routing* yang membentuk tabel *routing* hanya saat dibutuhkan dan protokol *proactive routing* yang melakukan pemeliharaan tabel *routing* secara berkala pada waktu tertentu. Pergerakan *node* pada VANETs bisa berubah setiap saat dan terbatas pada rute lalu lintas yang dapat ditentukan dari koordinat peta. Hal ini membuat setiap *node* akan terus memperbarui informasi dalam tabelnya sesuai informasi dari *node* lain. Perubahaan pergerakan pada VANETs menjadi salah satu permasalahan dalam pengiriman paket data sehingga dibutuhkan informasi jarak antar *node*, kecepatan dan *delay* transmisi [6]. Ilustrasi VANETs dapat dilihat pada Gambar 2.1.



Gambar 2.1 Ilustrasi VANETs [13]

Dalam Tugas Akhir ini, penulis akan mengimplementasikan *routing protocol* AODV yang dimodifikasi dan menguji performa protokol tersebut pada lingkungan VANETs.

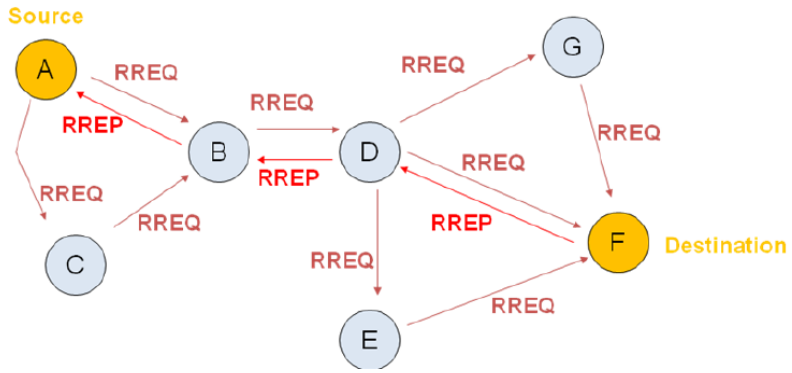
2.2 *Ad-hoc On demand Distance Vector* (AODV)

Ad-hoc On demand Distance Vector (AODV) adalah salah satu *routing* protokol yang termasuk dalam klasifikasi *reactive routing protocol*. Sebuah protokol yang hanya membuat sebuah rute saat dibutuhkan. AODV dikembangkan oleh C. E. Perkins, E.M. Belding-Royer dan S. Das pada RFC 3561.

Ciri utama dari AODV adalah menjaga *timer-based state* pada setiap *node* sesuai dengan penggunaan tabel *routing*. Tabel *routing*

akan kadaluarsa jika jarang digunakan. Ada dua tahapan dalam AODV yaitu *route discovery* dan r*oute maintenance. Route discovery* memiliki dua pesan yaitu berupa *Route Request* (RREQ) dan *Route Reply* (RREP). Sedangkan *Route maintenance* berupa *Route Error* (RERR).

AODV adalah sebuah metode *routing* pesan antar *node* yang memungkinkan *node-node* tersebut untuk melewatkan pesan melalui lingkungannya ke *node* yang tidak dapat dihubungi secara langsung. AODV melakukan ini dengan cara menemukan rute yang bisa dilalui oleh pesan. Selain itu AODV juga memastikan rute ini tidak mengandung perulangan (*loop*), menangani perubahan rute, dan membuat rute baru apabila terjadi *error* [7]. Ilustrasi pencarian rute oleh AODV dapat dilihat pada Gambar 2.2.



Gambar 2.2 Ilustrasi pencarian rute *routing protocol* AODV [14]

Pada setiap *node* yang menggunakan protokol AODV pasti memiliki sebuah *routing table* dengan *field* sebagai berikut:

- *Destination Address:* berisi alamat dari *node* tujuan.
- *Destination Sequence Number*: *sequence number* dari jalur komunikasi.
- *Next Hop*: alamat *node* yang akan meneruskan paket data.
- *Hop Count*: jumlah *hop* yang harus dilakukan agar paket dapat mencapai *node* tujuan.

- *Lifetime*: waktu dalam milidetik yang diperlukan *node* untuk menerima RREP.
- *Routing Flags*: status jalur. Terdapat tiga tipe status, yaitu *up* (valid), *down* (tidak valid) atau sedang diperbaiki.

Sebagai contoh proses *route discovery* dalam AODV, ilustrasi pada Gambar 2.2 menggambarkan bagaimana *source node*, yaitu *node* A mencari rute untuk menuju *destination node* yaitu *node* F. *Node* A akan membuat paket RREQ dan melakukan *broadcast* kepada semua *node* tetangganya (*neighbor node*). Jika *destination sequence number* yang terdapat pada paket RREQ sama atau lebih kecil dari yang ada pada *routing table* dan rute menuju *node* tujuan belum ditemukan, maka paket tersebut tidak akan dilanjutkan (*drop*). Jika *destination sequence number* pada RREQ lebih besar dibandingkan dengan yang terdapat pada *routing table*, maka *entry* pada *routing table* akan diperbarui dan paket tersebut akan diteruskan oleh *neighbor node* sekaligus membuat *reverse path* menuju *source node*. Paket RREQ akan diteruskan hingga mencapai *node* F. Kemudian, jika rute menuju *node* F sudah terbentuk di dalam *routing table* dan memiliki *routing flags* "*up*", maka *node* F akan mengirimkan paket RREP melalui rute tersebut menuju *node* .

Pada Tugas Akhir ini, penulis menggunakan *routing protocol* AODV yang akan diimplementasikan pada lingkungan VANETs dengan beberapa skenario.

2.3 *Network Simulator-2* (NS-2)

Network Simulator (NS) adalah suatu *interpreter* yang berorientasi objek, dan *discrete event-driven* yang dikembangkan oleh University of California Berkeley dan USC ISI sebagai bagian dari projek *Virtual INternet Testbed* (VINT). NS yang banyak dikenal dengan NS-2 (versi 2) menjadi salah satu *tool* yang sangat berguna untuk menunjukkan simulasi jaringan melibatkan *Local Area Network* (LAN), *Wide Area Network* (WAN), tapi fungsi dari *tool* ini telah berkembang selama beberapa tahun belakangan untuk

memasukkan jaringan nirkabel (*wireless*) dan juga jaringan *ad hoc* [8].

Pada Tugas Akhir ini, NS-2 digunakan untuk melakukan simulasi lingkungan VANETs menggunakan protokol AODV yang sudah dimodifikasi. *Trace file* yang dihasilkan oleh NS-2 juga digunakan untuk mengukur peforma *routing* protokol AODV yang sudah dimodifikasi..

2.3.1 Instalasi

NS-2 membutuhkan beberapa *package* yang harus sudah ter*install* sebelum memulai instalasi NS-2. Untuk meng*install dependency* yang dibutuhkan dapat dilakukan dengan *command* yang ditunjukkan pada Gambar 2.3

```
sudo apt-get install build-essential autoconf
automake libxmu-dev
```

Gambar 2.3 Perintah untuk meng*install dependency* NS-2

Setelah meng*install dependency* yang dibutuhkan, ekstrak *package* NS-2 dan ubah baris kode ke-137 pada *file* ls.h di *folder* linkstate menjadi seperti pada Gambar 2.4.

```
void eraseAll(){this->erase(baseMap::begin(),
baseMap::end()); }
```

Gambar 2.4 Baris kode yang diubah pada file ls.h

Install NS-2 dengan menjalankan perintah ./install pada folder NS-2.

2.3.2 *Trace File*

Trace file merupakan *file* hasil simulasi yang dilakukan oleh NS-2 dan berisikan informasi detail pengiriman paket data. *Trace file* digunakan untuk menganalisis performa *routing protocol* yang

disimulasikan. Detail penjelasan *trace file* ditunjukkan pada Tabel 2.1.

Tabel 2.1 Detail Penjelasan *Trace File* AODV

Kolom ke-	Penjelasan	Isi
1	*Event*	s : *sent* r : *received* f : *forwarded* D : *dropped*
2	*Time*	Waktu terjadinya *event*
3	ID *Node*	_x_ : dari 0 hingga banyak *node* pada topologi
4	*Layer*	AGT : *application* RTR : *routing* LL : *link layer* IFQ : *packet queue* MAC : *MAC* PHY : *physical*
5	*Flag*	--- : Tidak ada
6	*Sequence Number*	Nomor paket
7	*Packet Type*	AODV : paket *routing* AODV cbr : berkas paket CBR (*Constant Bit Rate*) RTS : *Request To Send* yang dihasilkan MAC 802.11 CTS : *Clear To Send* yang dihasilkan MAC 802.11 ACK : MAC ACK ARP : Paket *link layer address resolution protocol*
8	Ukuran	Ukuran paket pada *layer* saat itu
9	Detail MAC	[a b c d] a : perkiraan waktu paket b : alamat penerima

		c : alamat penerima d : IP header
10	*Flag*	-------- : Tidak ada
11	*Detail* IP *source, destination,* dan *nexthop*	[a:b c:d e f] a : IP *source node* b : *port source node* c : IP *destination node* (jika -1 berarti *broadcast*) d : *port destination node* e : IP *header* ttl f : IP *nexthop* (jika 0 berarti *node* 0 atau *broadcast*)

2.4 OpenStreetMap

OpenStreetMap (OSM) adalah sebuah proyek berbasis web untuk membuat peta dunia yang gratis dan terbuka, dibangun sepenuhnya oleh sukarelawan dengan melakukan survei menggunakan GPS, mendigitalisasi citra satelit dan mengumpulan serta membebaskan data geografis yang tersedia di publik. Melalui *Open Data Commons Open Database License 1.0,* kontributor OSM dapat memiliki, memodifikasi, dan membagikan data peta secara luas. Terdapat beragam jenis peta digital yang tersedia di internet, namun sebagian besar memiliki keterbatasan secara legal maupun teknis. Hal ini membuat masyarakat, pemerintah, peneliti dan akademisi, *inovator*, dan banyak pihak lainnya tidak dapat menggunakan data yang tersedia di dalam peta tersebut secara luas. Di sisi lain, baik peta dasar OSM maupun data yang tersedia di dalamnya dapat diunduh secara gratis dan terbuka, untuk kemudian digunakan untuk diredistribusikan kembali.

Di banyak tempat di dunia ini, terutama di daerah terpencil dan terbelakang secara ekonomi, tidak terdapat insentif komersil sama sekali bagi perusahaan pemetaan untuk mengembangkan data di tempat ini. OSM dapat menjadi jawaban di banyak tempat seperti ini,

baik itu pengembangan ekonomi, tata kota, kontinjensi bencana, maupun untuk berbagai tujuan lainnya [9].

Pada Tugas Akhir ini, penulis menggunakan data yang tersedia pada OSM untuk membuat skenario lalu lintas berdasarkan peta daerah di Surabaya. Peta yang diambil lalu digunakan untuk simulasi skenario *rea*l VANETs.

2.5 *Java OpenStreetMap Editor* (JOSM)

Java OpenStreetMap Editor (JOSM) adalah aplikasi untuk menyunting data yang didapatkan dari OpenStreetMap [10].

Pada Tugas Akhir ini, penulis menggunakan aplikasi ini untuk menyunting dan merapikan peta yang diunduh dari OpenStreetMap yaitu dengan menghilangkan dan menyambungkan jalan yang ada. Penyuntingan juga dilakukan dengan menghilangkan gedung – gedung yang ada di peta.

2.6 *Simulation of Urban Mobility* (SUMO)

Simulation of Urban Mobility (SUMO) merupakan paket simulasi lalu lintas yang bersifat *open-source* dimana pengembangannya dimulai pada tahun 2001. Dan semenjak itu SUMO telah berubah menjadi sebuah simulasi lalu lintas dengan kelengkapan fitur dan pemodelannya termasuk kemampuan jalannya jaringan untuk membaca *format* yang berbeda.

SUMO juga memungkinkan untuk mendefinisikan kendaraan dengan sifat tertentu seperti panjang kendaraan, kecepatan maksimum, percepatan dan perlambatannya. SUMO juga menyediakan pilihan bagi pengguna menentukan rute acak untuk kendaraan. Ada juga pilihan yang tersedia untuk model sistem transportasi umum, dimana setiap kendaraan datang dan berangkat sesuai dengan jadwal [11].

SUMO terdiri dari beberapa *tools* yang dapat membantu pembuatan simulasi lalu lintas pada tahap-tahap yang berbeda.

Berikut penjelasan fungsi *tools* yang digunakan dalam pembuatan Tugas Akhir ini:

- netgenerate
  netgenerate merupakan *tool* yang berfungsi untuk membuat peta berbentuk seperti *grid, spider*, dan bahkan *random network*. Sebelum proses netgenerate, pengguna dapat menentukan kecepatan maksimum jalan dan membuat *traffic light* pada peta. Hasil dari netgenerate ini berupa *file* dengan ekstensi .net.xml. Pada Tugas Akhir ini netgenerate digunakan untuk membuat peta skenario *grid*.
- netconvert
  netconvert merupakan program CLI yang berfungsi untuk melakukan konversi dari peta seperti OpenStreetMap menjadi format *native* SUMO. Pada Tugas Akhir ini penulis menggunakan netconvert untuk mengonversi peta dari OpenStreetMap.
- randomTrips.py
  Tool dalam SUMO untuk membuat rute acak yang akan dilalui oleh kendaraan dalam simulasi.
- duarouter
  Tool dalam SUMO untuk melakukan perhitungan rute berdasarkan definisi yang diberikan dan memperbaiki kerusakan rute.
- sumo
  Program yang melakukan simulasi lalu lintas berdasarkan data-data yang didapatkan dari netgenerate (skenario *grid*) atau netconvert dari randomTrips.py. Hasil simulasi dapat di-*export* ke sebuah *file* untuk dikonversi menjadi format lain.
- sumo-gui
  GUI untuk melihat simulasi yang dilakukan oleh SUMO secara grafis.
- traceExporter.py
  Tool yang bertujuan untuk mengonversi *output* dari sumo menjadi format yang dapat digunakan pada *simulator* lain.

Pada Tugas Akhir ini penulis menggunakan traceExporter.py untuk mengonversi data menjadi format .tcl yang dapat digunakan pada NS-2

Pada Tugas Akhir ini, penulis menggunakan SUMO untuk menghasilkan skenario VANETs, peta area simulasi, dan pergerakan *node* sehingga menyerupai keadaan lalu lintas yang sebenarnya. Untuk setiap skenario VANETs yang dibuat menggunakan SUMO, akan dihasilkan pergerakan *node* yang acak sehingga setiap skenario memiliki pergerakan yang berbeda.

2.7 AWK

AWK adalah bahasa pemrograman yang digunakan untuk melakukan *text processing* dan ekstraksi data [12]. AWK merupakan sebuah program filter untuk teks, seperti halnya perintah grep pada terminal linux. AWK dapat digunakan untuk mencari bentuk / model dalam sebuah berkas teks ke dalam bentuk teks lain. AWK dapat juga digunakan untuk melakukan proses aritmatika seperti yang dilakukan oleh perintah expr. AWK sama halnya seperti bahasa shell atau C yang memiliki karakteristik yaitu sebagai *tool* yang cocok untuk *jobs* juga sebagai pelengkap untuk *filter* standar.

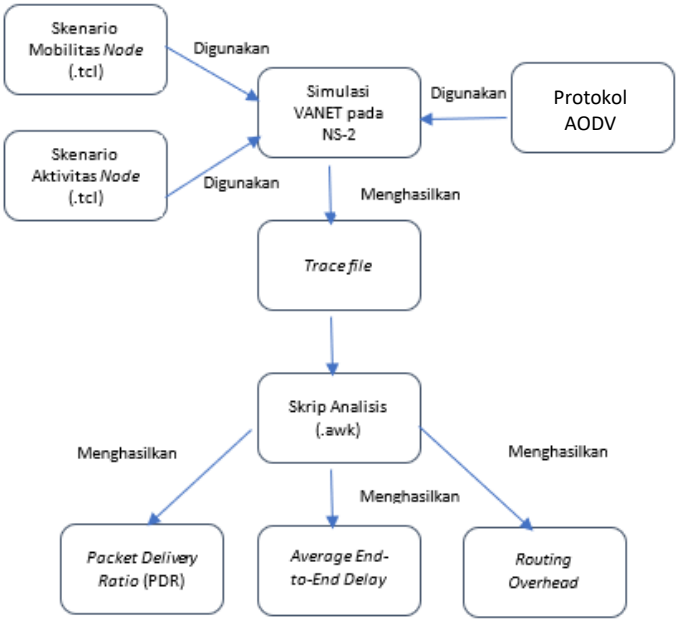
Pada Tugas Akhir ini, AWK digunakan untuk membuat script menghitung *Packet Delivery Ratio* (PDR*), Routing Overhead* (RO), *Forwarded Route Request* (RREQ F), dan *End-to-End Delay* dari *trace file* NS2.

BAB III
PERANCANGAN

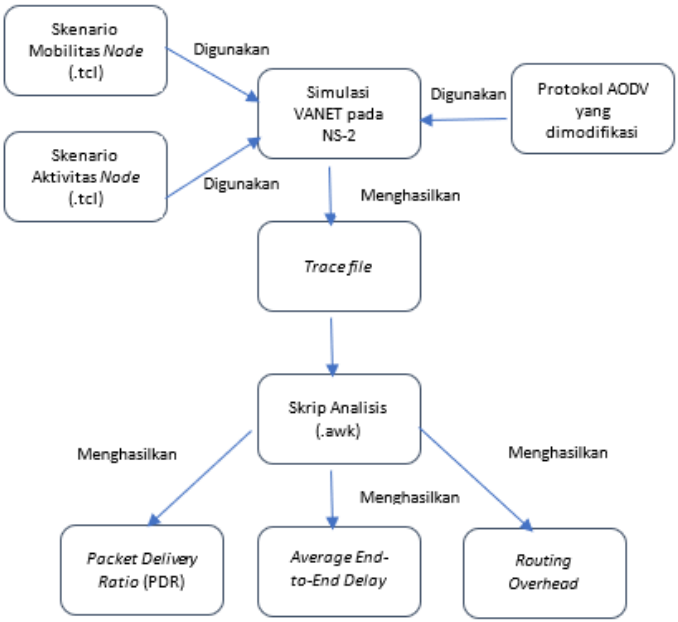
Perancangan merupakan bagian penting dari pembuatan sistem secara teknis sehingga bab ini secara khusus menjelaskan perancangan sistem yang dibuat dalam Tugas Akhir. Berawal dari deskripsi umum sistem hingga perancangan skenario, alur dan implementasinya.

3.1 Deskripsi Umum

Pada Tugas Akhir ini akan diimplementasikan *routing protocol* AODV dengan memodifikasi pada bagian proses *route discovery* yang dijalankan pada simulator NS-2. Diagram dari rancangan simulasi dari AODV asli dan AODV modifikasi dapat dilihat pada Gambar 3.1 dan Gambar 3.2.



Gambar 3.1 Diagram Rancangan Simulasi AODV Asli



Gambar 3.2 Diagram Rancangan Simulasi AODV Modifikasi

Modifikasi akan diawali dengan pencarian jumlah tetangga setiap *node* perantara (*one-hop node)*. Jumlah tetangga tiap *node* akan didapatkan dengan menggunakan HELLO *messages* yang terdapat pada AODV. Setelah jumlah tetangga setiap *node* didapatkan, maka modifikasi dilanjutkan untuk melakukan penyeleksian *forwarding node* yang bertugas untuk *rebroadcast* paket *Route Request* (RREQ) yang akan diteruskan. Hal tersebut dilakukan dengan cara menyeleksi *node – node* mana saja yang mempunyai jumlah tetangga lebih dari nilai *threshold* yang sudah ditentukan. Jika *node* tersebut mempunyai jumlah tetangga lebih dari *threshold*, maka *node* tersebut akan meneruskan paket RREQ, namun jika sebaliknya, maka *node* tersebut tidak akan meneruskan paket RREQ atau paket akan di-*drop*.

Modifikasi yang telah dilakukan akan disimulasikan pada NS-2 dengan peta berbentuk *grid* dan peta *real* pada lingkungan lalu lintas di kota Surabaya. Pembuatan kedua peta tersebut menggunakan bantuan *tools* SUMO. Simulasi tersebut akan memberikan hasil *trace file* yang kemudian dianalisis menggunakan skrip AWK untuk mendapatkan *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Forwarded Route Request* (RREQ F), dan *End-to-End Delay* (E2E). Analisis tersebut dapat mengukur performa *routing protocol* AODV yang telah dimodifikasi dibandingkan dengan *routing protocol* AODV sebelum dimodifikasi. Analisis ini digunakan untuk mengukur tingkat reliabilitas pengiriman data antara protokol AODV dengan protokol AODV yang dimodifikasi. Daftar istilah yang sering digunakan pada buku Tugas Akhir ini dapat dilihat pada Tabel 3.1.

Tabel 3.1 Daftar Istilah

No.	**Istilah**	**Penjelasan**
1	AODV	Singkatan dari *Ad hoc On-demand Distance Vector*. Protokol yang digunakan pada Tugas Akhir ini.
2	PDR	*Packet Delivery Ratio*. Salah satu metrik analisis yang diukur. Berupa rasio jumlah pengiriman paket yang terkirim.
3	E2E	*Average End-to-End Delay*. Jeda waktu yang diukur saat paket terkirim.
4	RO	*Routing Overhead*. Jumlah *control packet* yang terkirim
5	RREQ	*Route Request*. Paket *request* pada AODV yang dikirim untuk mendapatkan rute
6	RREP	*Route Reply*. Paket *reply* pada AODV yang dikirim ke *node* sumber melalui rute yang sudah terbuat.
7	*Threshold*	Batas nilai yang dijadikan sebagai acuan

3.2 Perancangan Skenario Mobilitas

Perancangan skenario mobilitas dimulai dengan membuat area simulasi, pergerakan *node*, dan implementasi pergerakan *node*. Dalam Tugas Akhir ini, terdapat dua macam area simulasi yang akan digunakan yaitu peta *grid* dan *real*. Peta *grid* yang dimaksud adalah bentuk jalan berpetak – petak sebagai contoh jalan berpotongan yang sederhana. Peta *grid* digunakan sebagai simulasi awal VANETs karena lebih stabil. Peta *grid* didapatkan dengan menentukan panjang dan jumlah petak area menggunakan SUMO. Sedangkan yang dimaksud peta *real* adalah peta asli / nyata yang digunakan sebagai area simulasi. Peta *real* didapatkan dengan mengambil daerah yang diinginkan sebagai area simulasi menggunakan OpenStreetMap. Pada Tugas Akhir ini, peta *real* yang diambil penulis adalah salah satu area di Surabaya.

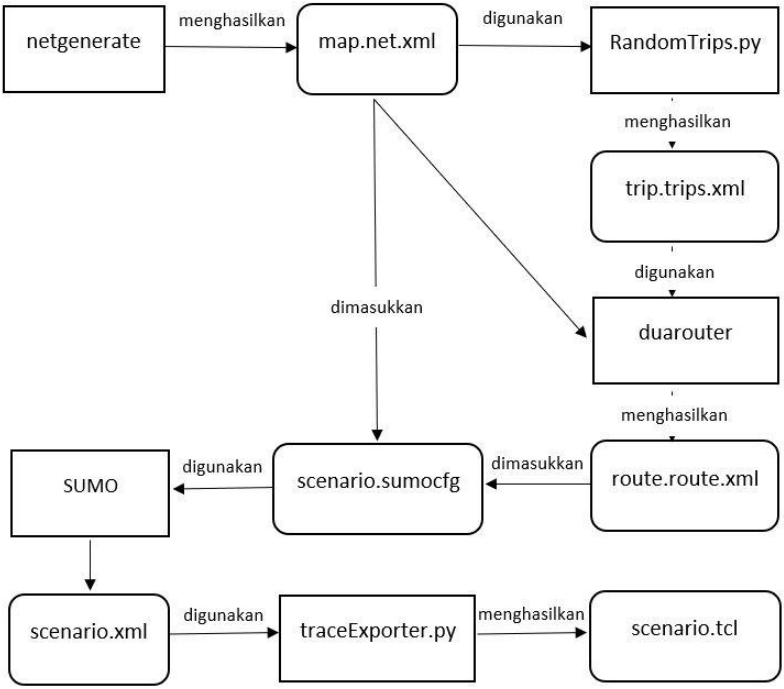
3.2.1 Perancangan Skenario *Grid*

Perancangan skenario mobilitas *grid* diawali dengan merancang luas area peta *grid* yang dibutuhkan. Luas area tersebut bisa didapatkan dengan cara menentukan terlebih dahulu jumlah titik persimpangan yang diinginkan, sehingga dari jumlah persimpangan tersebut dapat diketahui berapa banyak peta yang dibutuhkan. Dengan mengetahui jumlah petak yang dibutuhkan, dapat ditentukan panjang tiap petak sehingga mendapatkan luas area yang dibutuhkan yaitu 1300 m x 1300 m. Dengan 4 titip persimpangan, maka akan didapatkan 9 petak dan panjang tiap peta adalah 400m.

Peta *grid* yang telah ditentukan luasnya tersebut kemudian dibuat dengan menggunakan *tools* SUMO yaitu netgenerate. Selain titik persimpangan dan panjang tiap petak *grid*, dibutuhkan juga pengaturan kecepatan kendaraan menggunakan *tools* tersebut. Peta *grid* yang dihasilkan oleh netgenerate akan memiliki ekstensi .net.xml. Peta *grid* ini kemudian digunakan untuk membuat

pergerakan *node* dengan *tools* SUMO yaitu menggunakan *tools* randomTrips dan duarouter.

Skenario mobilitas *grid* dihasilkan dengan menggabungkan *file* peta *grid* dan *file* pergerakan *node* yang telah dibuat. Penggabungan tersebut menghasilkan *file* dengan ekstensi .xml. Selanjutnya, untuk dapat diterapkan pada NS-2, *file* skenario mobilitas *grid* yang berekstensi .xml dikonversi ke dalam bentuk *file* .tcl. Konversi ini dilakukan menggunakan *tool* traceExporter. Alur pembuatan skenario *grid* dapat dilihat pada Gambar 3.3.

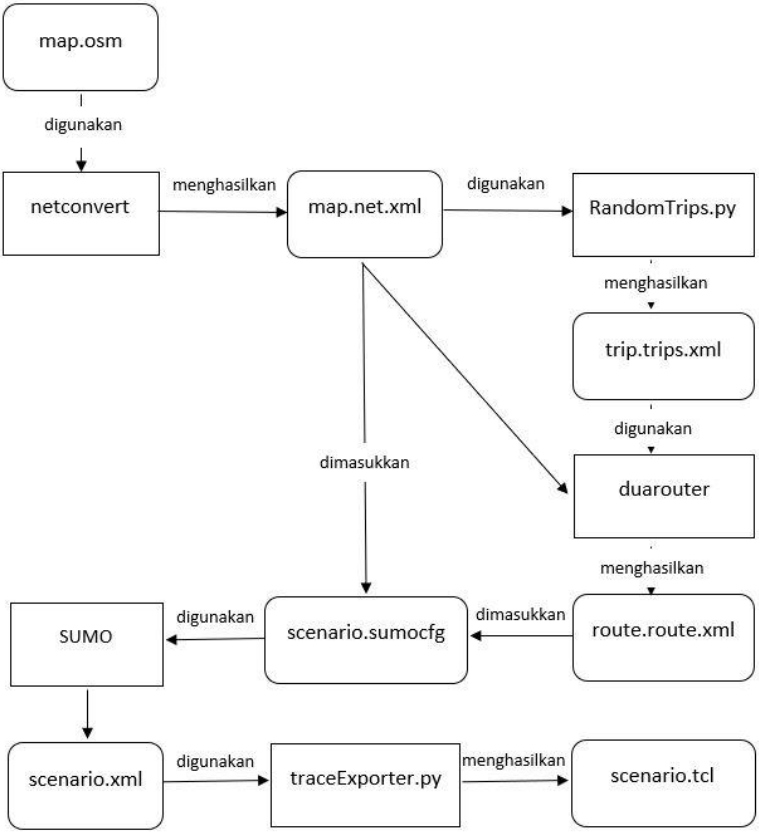


Gambar *3.3* Alur Pembuatan Skenario *Grid*

3.2.2 Perancangan Skenario *Real*

Perancangan skenario mobilitas *real* diawali dengan memilih area yang akan dijadikan simulasi. Pada Tugas Akhir ini, digunakan peta dari OpenStreetMap untuk mengambil area yang dijadikan model simulasi. Setelah memilih area, dilakukan pengunduhan dengan menggunakan fitur *export* yang telah disediakan oleh OpenStreetMap. Peta hasil *export* tersebut memiliki ekstensi .osm.

Setelah mendapatkan peta area yang akan dijadikan simulasi, peta tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .net.xml menggunakan *tools* SUMO yaitu netconvert. Tahap berikutnya memiliki tahapan yang sama seperti merancang skenario *grid*, yaitu membuat pergerakan *node* menggunakan randomTrips dan duarouter. Kemudian dilakukan penggabungan *file* peta *real* yang sudah dikonversi ke dalam *file* dengan ekstensi .net.xml dan *file* pergerakan *node* yang sudah dibuat sebelumnya. Hasil dari penggabungan tersebut merupakan *file* skenario berkekstensi .xml. *File* yang dihasilkan tersebut dikonversi ke dalam bentuk *file* dengan ekstensi .tcl agar dapat diterapkan pada NS-2. Alur pembuatan skenario *real* dapat dilihat pada Gambar 3.4.



Gambar 3.4 Alur Pembuatan Skenario *Real*

3.3 Perancangan Modifikasi *Routing Protocol* AODV

Protokol AODV yang diajukan pada Tugas Akhir ini merupakan modifikasi dari protokol AODV yang mengubah mekanisme *route discovery* pada protokol tersebut. Pada protokol AODV, mekanisme pencarian *node* untuk pengiriman ulang (*rebroadcast*) paket RREQ langsung dikirim begitu saja, maka pada AODV yang dimodifikasi ini akan ada proses penyeleksian *node*.

Seleksi *node* dilakukan dengan cara *node* sumber mengetahui jumlah tetangga yang dimiliki *node* perantara (*one-hop node*). Selanjutnya, dilakukan perbandingan menggunakan *threshold* yang sudah ditentukan sebelumnya untuk pengiriman RREQ. Apabila jumlah tetangga *one-hop node* tersebut lebih besar atau sama dengan dengan *threshold* yang sudah ditentukan, maka pengiriman RREQ akan dilanjutkan. Namun sebaliknya, apabila jumlahnya kurang dari *threshold*, maka paket akan di*drop*. Jika sudah mencapai *node* tujuan, *node* akan mengembalikannya dengan paket RREP. Rute yang dilalui paket RREP adalah rute dengan pembatasan *forwarding node* yang sudah dilakukan sebelumnya. Rute tersebut tidak melakukan *rebroadcast* RREQ ke semua *node* dan paket RREQ hanya melewati *node* – *node* terpilih untuk sampai ke *node* tujuan. Karena beberapa paket RREQ di-drop, maka kemacetan dan tabrakan paket pada jaringan akan lebih rendah sehingga bisa menaikkan PDR.

3.3.1 Perancangan Penghitungan Jumlah *Node* Tetangga untuk Setiap *Node*

Ada beberapa cara untuk dapat mengetahui jumlah tetangga yang ada di sekitar *node*. Pada Tugas Akhir ini, penghitungan jumlah tetangga dilakukan dengan memanfaatkan HELLO *messages* yang ada pada protokol AODV. HELLO *packets* akan mengirimkan HELLO *messages* secara terus menerus untuk memberikan informasi kepada *node* tersebut mengenai tetangga yang ada di sekitarnya. Setiap *node* yang menerima HELLO *messages* dari *node* lainnya, sudah dipastikan menjadi tetangga *node* tersebut. Dengan begitu, jumlah tetangga dapat dihitung dari setiap *node* yang mengirimkan HELLO *messages*. Modifikasi akan dilakukan dengan memanfaatkan fungsi `addNeighbor()` pada file node.cc yang terletak di dalam *folder* common/ns-2.35 dan `recvHello()` pada file aodv.cc yang terletak di dalam *folder* ns-2.35/aodv. *Pseudocode* untuk penghitungan jumlah *node* tetangga dapat dilihat pada Gambar 3.5.

```
flag = false

while neighbors is exist
      if neighbors->id = new_neighbor->id
            flag = true
            break
      else
            neighbors = neighbors->next
      end if
if flag = false
      new_neighbor->next = neighbors
      neighbors = new_neighbor
end if
```

Gambar 3.5 Pseudocode Penghitungan Jumlah Node Tetangga

3.3.2 Perancangan Pemilihan *Forwarding Node*

Setelah melakukan penghitungan jumlah *node* tetangga, akan dilakukan pemilihan *forwarding node*, yaitu *node* yang berhak untuk melakukan *rebroadcast* paket RREQ. Penentuan *forwarding node* dilakukan dengan cara membandingkan jumlah *node* tetangga dengan *threshold* atau batas nilai yang sudah ditentukan. Apabila jumlah tetangga yang dimiliki *node* tersebut melebihi *threshold*, maka *node* tersebut berhak melakukan *rebroadcast*, jika sebaliknya maka paket akan di-*drop*. Langkah tersebut dilakukan untuk mengurangi jumlah paket RREQ yang dikirim sehingga dapat mengurangi kemacetan yang terjadi pada jaringan. *Pseudocode* untuk pemilihan *forwarding node* dapat dilihat pada Gambar 3.6.

```
count = 0
while neighbors is exist
      increment count
      if neighbors->next is exist
            neighbors = neighbors->next
      else
            break
      end if

if count < threshold
      drop RREQ packet
end if
```

Gambar 3.6 Pseudocode Pemilihan *Forwarding Node*

3.4 Perancangan Simulasi pada NS-2

Simulasi VANETs pada NS-2 dilakukan dengan menggabungkan *file* skenario yang telah dibuat menggunakan SUMO dan *file* skrip dengan ekstensi .tcl yang berisikan konfigurasi lingkungan simulasi.

Kode yang diubah diantaranya adalah pencarian jumlah *node* tetangga pada *file* node.cc dan perbandingan dengan *threshold* pada *file* aodv.cc. Pada saat simulasi NS-2 dijalankan, maka *routing protocol* AODV akan menyeleksi *node* mana saja yang berhak melakukan *rebroadcast* paket RREQ.

3.5 Perancangan Metrik Analisis

Berikut ini merupakan parameter – parameter yang akan dianalisis pada Tugas Akhir ini untuk dapat membandingkan performa dari *routing protocol* AODV yang asli dengan AODV yang telah dimodifikasi:

3.5.1 *Packet Delivery Ratio* (PDR)

Packet delivery ratio merupakan perbandingan dari jumlah paket data yang dikirim dengan paket data yang diterima. PDR dapat menunjukkan keberhasilan paket yang dikirimkan. Semakin tinggi PDR artinya semakin berhasil pengiriman paket yang dilakukan Rumus untuk menghitung PDR dapat dilihat pada persamaan 3.1.

$$PDR = \frac{received}{sent} x\ 100\ \% \qquad (3.1)$$

Keterangan:

PDR = *Packet Delivery Ratio*

received = banyak paket data yang diterima

sent = banyak paket data yang dikirimkan

3.5.2 *Average End-to-End Delay* (E2E)

Average End-to-End Delay dihitung berdasarkan rata-rata *delay* antara waktu paket data diterima dan waktu paket dikirimkan dalam satuan detik. Delay tiap paket didapat dari rentang waktu antara *node* asal saat mengirimkan paket dan *node* tujuan menerima paket. Delay tiap paket tersebut semua dijumlahkan dan dibagi dengan jumlah paket yang berhasil diterima, maka akan didapatkan rata – rata E2E, yang dapat dihitung dengan persamaan 3.2.

$$E2E = \frac{\sum_{m=1}^{recvnum} CBRRecvTime - CBRSentTime}{recvnum} \qquad (3.2)$$

Keterangan:

E2E = *End-to-End Delay*

CBRRecvTime = Waktu *node* asal mengirimkan paket

CBRSentTime = Waktu *node* tujuan menerima paket

recvnum = Jumlah paket yang berhasil diterima

3.5.3 *Routing Overhead* (RO)

Routing Overhead adalah jumlah paket kontrol *routing* yang ditransmisikan per data paket ke *node* tujuan selama simulasi terjadi. *Routing Overhead* didapatkan dengan menjumlahkan semua paket kontrol *routing* yang ditransmisikan, baik itu paket *route request* (RREQ), *route reply* (RREP), maupun *route error* (RERR).. Perhitungan *Routing Overhead* dapat dilihat dengan persamaan 3.3.

$$RO = \sum_{m=1}^{sentnum} packet\ sent \quad (3.3)$$

3.5.4 *Forwarded Route Request* (RREQ F)

Forwarded Route Request adalah jumlah paket kontrol *route request* yang di*forward* per data paket ke *node* tujuan selama simulasi terjadi. RREQ F didapatkan dengan menjumlahkan semua paket control *routing* yang ditransmisikan khususnya *route request* bagian *forwarding* (RREQ F). Perhitungan RREQ F dapat dilihat dengan persamaan 3.4

$$Forwarded\ Route\ Request = \sum_{n=1}^{rreqsent} packet\ sent \quad (3.4)$$

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan yang sudah dilakukan pada bab sebelumnya. Implementasi berupa kode sumber untuk membangun program.

4.1 Implementasi Skenario Mobilitas

Implementasi skenario mobilitas VANETs dibagi menjadi dua, yaitu skenario *grid* yang menggunakan peta jalan berpetak dan skenario *real* yang menggunakan peta hasil pengambilan suatu area di kota Surabaya.

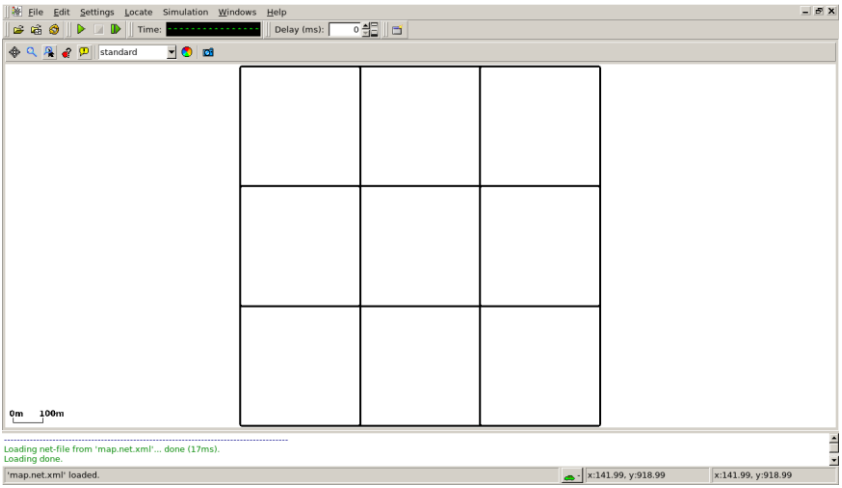
4.1.1 Skenario *Grid*

Dalam mengimplementasikan skenario *grid*, SUMO menyediakan *tools* untuk membuat peta *grid* yaitu netgenerate. Pada Tugas Akhir ini, penulis membuat peta *grid* dengan luas 1300 m x 1300 m yang terdiri dari titik persimpangan antara jalan vertikal dan jalan horisontal sebanyak 4 titik x 4 titik. Dengan jumlah titik persimpangan sebanyak 4 titik tersebut, makat terbentuk 9 buah petak. Sehingga untuk mencapai luas area sebesar 1300 m x 1300 m dibutuhkan luas per petak sebesar 400 m x 400 m. Berikut perintah netgenerate untuk membuat peta tersebut dengan kecepatan *default* kendaraan sebesar 20m/s dapat dilihat pada Gambar 4.1.

```
netgenerate --grid --grid.number=4 --
grid.length=400 --default.speed=20 --
tls.guess=1 --output-file=map.net.xml
```

Gambar 4.1 Perintah netgenerate

Setelah itu akan didapat *file* peta berekstensi .xml. Gambar hasil peta yang telah dibuat dengan netgenerate dapat dilihat pada Gambar 4.2.



Gambar 4.2 Hasil *Generate* Peta *Grid*

Setelah peta terbentuk, maka dilakukan pembuatan *node* dan pergerakan *node* dengan menentukan titik awal dan titik akhir setiap *node* seacara random menggunakan *tools* randomTrips yang terdapat di SUMO. Perintah penggunaan *tools* randomTrips untuk membuat *node* sebanyak n *node* dengan pergerakannya dapat dilihat pada Gambar 4.3.

```
python $SUMO_HOME/tools/randomTrips.py -n
map.net.xml -e 58 -l --trip-
attributes="departLane=\"best\"
departSpeed=\"max\"
departPos=\"random_free\"" -o trip.trips.xml
```

Gambar 4.3 Perintah randomTrips

Selanjutnya dibuatkan rute yang digunakan kendaraan untuk mencapai tujuan dari *file* hasil sebelumnya menggunakan *tools* duarouter. Perintah penggunaan *tools* duarouter dapat dilihat pada Gambar 4.4.

```
duarouter -n map.net.xml -t trip.trips.xml -
o route.rou.xml --ignore-errors --repair
```

Gambar 4.4 Perintah duarouter

Ketika menggunakan *tools* duarouter, SUMO memastikan bahwa jalur untuk *node-node* yang di*generate* tidak akan melenceng dari jalur peta yang sudah di*generate* menggunakan *tools* randomTrips. Selanjutnya untuk menjadikan peta dan pergerakan *node* yang telah di*generate* menjadi sebuah skenario dalam bentuk *file* berekstensi .xml, dibutuhkan sebuah *file* skrip dengan ekstensi .sumocfg untuk menggabungkan *file* peta dan rute pergerakan *node*. Isi dari *file* skrip .sumocfg dapat dilihat pada Gambar 4.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration
xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance"
xsi:noNamespaceSchemaLocation="http://sumo.
dlr.de/xsd/sumoConfiguration.xsd">
   <input>
      <net-file value="map.net.xml"/>
      <route-files value="routes.rou.xml"/>
   </input>
   <time>
      <begin value="0"/>
      <end value="200"/>
   </time>
</configuration>
```

Gambar 4.5 File Skrip .sumocfg

File .sumocfg disimpan dalam direktori yang sama dengan *file* peta dan *file* rute pergerakan *node.* Untuk percobaan sebelum dikonversi, *file* .sumocfg dapat dibuka dengan menggunakan *tools* sumo-gui. Kemudian buat *file* skenario dalam bentuk *file* .xml dari sebuah *file* skrip berekstensi .sumocfg menggunakan *tools* SUMO. Perintah untuk menggunakan *tools* SUMO dapat dilihat pada Gambar 4.6.

```
sumo -c file.sumocfg --fcd-output
scenario.xml
```

Gambar 4.6 Perintah SUMO untuk membuat skenario .xml

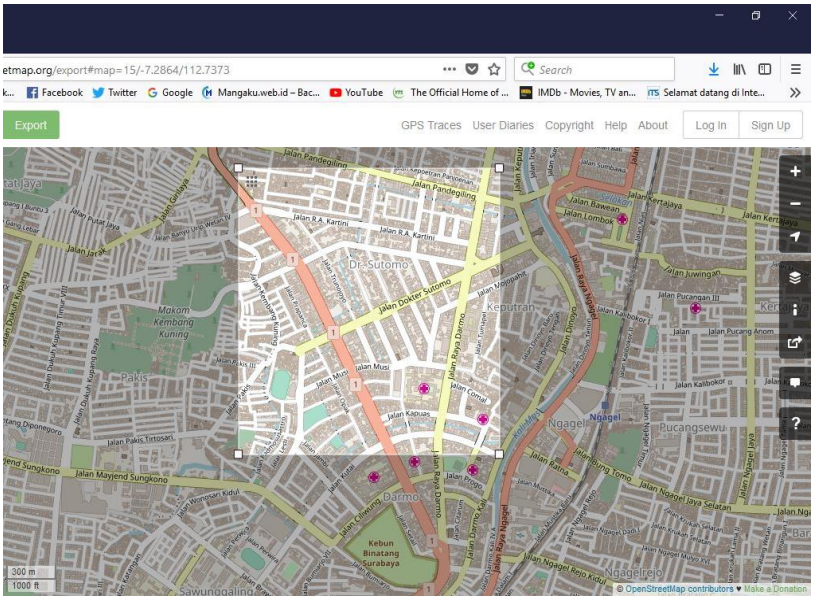
File skenario berekstensi .xml selanjutnya dikonversi ke dalam bentuk *file* berekstensi .tcl agar dapat disimulasikan menggunakan NS-2. *Tools* yang digunakn untuk melakukan konversi ini adalah traceExporter. Perintah untuk menggunakan traceExporter dapat dilihat pada Gambar 4.7.

```
python $SUMO_HOME/tools/traceExporter.py --
fcd-input=scenario.xml --ns2mobility-
output=scenario.tcl
```

Gambar 4.7 Perintah traceExporter

4.1.2 Skenario *Real*

Dalam mengimplementasikan skenario *real*, langkah pertama adalah menentukan area yang akan dijadikan area simulasi. Pada Tugas Akhir ini penulis mengambil area jalan sekitar Jl. Dr. Soetomo Surabaya. Setelah menentukan area simulasi, ekspor data peta dari OpenStreetMap seperti yang ditunjukkan pada Gambar 4.8.



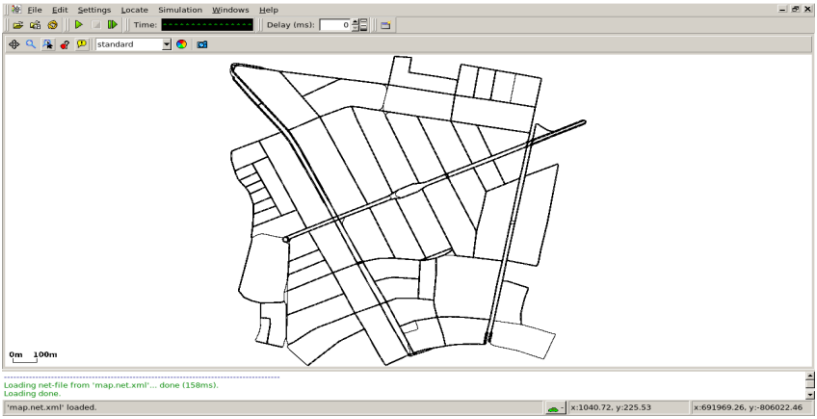
Gambar 4.8 Ekspor Peta dari OpenStreetMap

File hasil ekspor dari OpenStreetMap tersebut adalah *file* peta dengan ekstensi .osm. Kemudian konversi *file* .osm tersebut menjadi peta dalam bentuk *file* berekstensi .xml menggunakan *tools* netconvert dari SUMO. Perintah untuk menggunakan netconvert dapat dilihat pada Gambar 4.9.

```
netconvert --osm-files map.osm --output-
file map.net.xml
```

Gambar 4.9 Perintah netconvert

Hasil konversi peta dari *file* berekstensi .osm menjadi *file* berekstensi .xml dapat dilihat menggunakan *tools* sumo-gui seperti yang ditunjukkan pada Gambar 4.10.



Gambar 4.10 Hasil Konversi Peta *Real*

Langkah selanjutnya sama dengan ketika membuat skenario mobilitas *grid*, yaitu membuat *node* asal dan *node* tujuan menggunakan *tool* randomTrips. Lalu membuat rute *node* untuk sampai ke tujuan menggunakan *tool* duarouter. Kemudian membuat *file* skenario berekstensi .xml menggunakan *tool* SUMO dengan bantuan *file* skrip berekstensi .sumocfg. Selanjutnya dilakukan

konversi *file* skenario berekstensi .tcl untuk dapat disimulasikan pada NS-2 menggunakan *tool* traceExporter. Perintah untuk menggunakan *tools* tersebut sama dengan ketika membuat skenario *grid* di atas.

4.2 Implementasi Modifikasi pada *Routing Protocol* AODV untuk Menentukan *Forwarding Node*

Pada Tugas Akhir ini dilakukan modifikasi pada *routing protocol* AODV agar dapat mengurangi jumlah *forwarding node*, yaitu *node* yang bertugas untuk melakukan *rebroadcast* paket RREQ. Hal tersebut dilakukan dengan cara memilih *forwarding node* berdasarkan jumlah tetangga *node* tersebut dengan *threshold* yang sudah ditentukan, sehingga dapat dilihat peningkatan performa pada *routing* AODV yang telah dimodifikasi.

Implementasi modifikasi routing protocol AODV ini dibagi menjadi 2 bagian yaitu:

- Implementasi Penghitungan Jumlah *Node* Tetangga
- Implementasi Pemilihan *Forwarding Node*

Kode implementasi dari routing protocol AODV pada NS-2 versi 2.35 berada pada direktori ns-2.35/aodv. Pada direktori tersebut terdapat beberapa file diantaranya seperti aodv.cc, aodv.h dan sebagainya. Pada Tugas Akhir ini, penulis memodifikasi *file* node.cc yang terdapat dalam *folder* ns-2.35/common untuk menghitung jumlah *node* tetangga dan *file* aodv.cc yang ada di dalam *folder* ns-2.35/aodv untuk menentukan *forwarding node*. Pada bagian ini penulis akan menjelaskan langkah – langkah dalam mengimplementasikan modifikasi *routing protocol* AODV untuk mengurangi jumlah *forwarding node* yang melakukan *rebroadcast*.

4.2.1 Implementasi Penghitungan Jumlah *Node* Tetangga

Langkah awal yang dilakukan untuk menghitung jumlah tetangga seperti yang telah dirancang pada subbab 3.3.1 adalah dengan cara menangkap pesan HELLO *messages* dari *node* yang mengirimkannya.

Node yang mengirimkan HELLO *messages* sudah dapat dipastikan berada di sekitar *node* tersebut dan berada di *range* transmisi dari *node* yang sedang dieksekusi. Secara *default*, AODV menginformasikan siapa saja *node* tetangga yang ada di sekitar *node* tersebut. Tetapi, apabila *node* tetangga menjauh kemudian mendekat lagi, HELLO akan menghitungnya lagi sebagai tetangga yang baru. Modifikasi dilakukan untuk mengecek apakah *node* tetangga pernah dimasukkan ke dalam list apa tidak. Tugas Akhir ini memanfaatkan fungsi `addNeighbor()` pada kode sumber node.cc yang terdapat dalam folder ns2.35/common. Untuk potongan kode tersebut bisa dilihat pada Gambar 4.11. Kode selengkapnya dapat dilihat di lampiran A1.

```
while(my_neighbor_list){
      if(my_neighbor_list->nodeid ==
      neighbor->nodeid()){
      flag = 1;
      break;}
      else{
            my_neighbor_list=my_neighbor_list
      ->next;
      }
}
if(flag == 0){
      neighbor_list_node* nlistItem =
      (neighbor_list_node*)malloc(sizeof(neig
      hbor_list_node));
      nlistItem->nodeid = neighbor->nodeid();
      nlistItem->next = neighbor_list_;
      neighbor_list_=nlistItem;
}
```

Gambar 4.11 Potongan Kode Memasukkan *Node* Tetangga ke dalam *List*

Fungsi tersebut akan dipanggil pada fungsi `recvHello()` yang terdapat pada kode sumber aodv.cc dalam folder ns2.35/aodv. Pada fungsi `recvHello()`, HELLO *messages* akan diterima, kemudian fungsi `addNeighbor()` akan dipanggil untuk memasukkannya ke dalam list. Untuk potongan kode tersebut dapat dilihat pada Gambar 4.12.

```
if (nb == 0)
{
    sender_node->addNeighbor(receiver_node);
    receiver_node->addNeighbor(sender_node);
    nb_insert(rp->rp_dst);
}
```

Gambar 4.12 Potongan Kode Memanggil fungsi addNeighbor ()

Selanjutnya, untuk melakukan penghitungan jumlah *node*, dilakukan pengecekan pada neighbor list. Kode diimplementasikan pada fungsi recvRequest() karena akan dilakukan perbandingan *threshold* pada fungsi tersebut. Potongan kode dapat dilihat pada Gambar 4.13. Kode implementasi ini diletakkan pada lampiran A.1 Kode Fungsi addNeighbor() dan A.2 Kode Fungsi recvHello().

```
Node* m_node =
Node::get_node_by_address(this->addr());
Neighbor_list_node* my_mobile_neighbor_list;
my_mobile_neighbor_list = m_node-
>neighbor_list_;
int count = 0;

while(my_mobile_neighbor_list){
      count++;

     if(my_mobile_neighbor_list->next){

           my_mobile_neighbor_list=my_mobile
           _neighbor_list->next;
      }
      else{
           break;
      }
}
```

Gambar 4.13 Potongan Kode Menghitung Jumlah *Node* Tetangga

4.2.2 Implementasi Pemilihan *Forwarding Node*

Pada tahap selanjutnya setelah dapat mengetahui jumlah tetangga, langkah yang harus dilakukan selanjutnya adalah pemilihan *forwarding node* untuk melanjutkan paket RREQ.

Penghitungan ini dilakukan pada fungsi `recvRequest()` yang terletak pada kode sumber aodv.cc yang terletak pada ns2.35/aodv. Jumlah tetangga yang sudah ditemukan, disimpan dalam variabel *count*. Jumlah tetangga kemudian akan dibandingkan dengan *threshold* yang sudah ditentukan dan bukan merupakan *node* sumber. Nilai *threshold* didapatkan dari percobaan dari angka 1 sampai batas maksimal *node* yang dilakukan selama 10 kali dan ditentukan dari nilai yang paling sering muncul dalam percobaan tersebut. Apabila

jumlah tetangga kurang dari *threshold* yang ditentukan dan bukan *node* sumber, maka paket RREQ akan di*drop*. Potongan kode untuk proses seleksi *forwarding node* dapat dilihat pada Gambar 4.14.

```
if (count < threshold && rq->rq_dst != 
index){
      Packet::free(p);
      return;
}
```

Gambar 4.14 Potongan Kode Penyeleksian *Forwarding Node*

Dengan proses penyeleksian *node* mana saja yang dapat melanjutkan paket RREQ sudah dapat mengurangi jumlah paket *routing overhead* untuk paket RREQ. Kode implementasi ini diletakkan pada lampiran A.3 Kode Fungsi recvRequest().

4.3 Implementasi Simulasi pada NS-2

Implementasi simulasi VANETs diawali dengan pendeskripsian lingkungan simulasi pada sebuah *file* tcl. *File* ini berisikan konfigurasi setiap *node* dan langkah-langkah yang dilakukan selama simulasi. Potongan konfigurasi lingkungan simulasi dapat dilihat pada Gambar 4.15.

```
set val(chan)    Channel/WirelessChannel
set val(prop)    Propagation/TwoRayGround
set val(netif)   Phy/WirelessPhy
set val(mac)     Mac/802_11
set val(ifq)     Queue/DropTail/PriQueue
set val(ll)      LL
set val(ant)     Antenna/OmniAntenna
set opt(x)       1500
set opt(y)       1500
set val(ifqlen)  1000
set val(nn)      60
set val(seed)    1.0
set val(adhocRouting) AODV
set val(stop)    200
set val(cp)      "cbr60.txt"
set val(sc)      "scenario1.txt"
```

Gambar 4.15 Implementasi Simulasi NS-2

Pada konfigurasi dilakukan pemanggilan terhadap *file traffic* yang berisikan konfigurasi *node* asal, *node* tujuan, pengiriman paket, serta *file* skenario yang berisi pergerakan *node* yang telah di*generate* oleh SUMO. Kode implementasi pada NS-2 dapat dilihat pada lampiran A.4 Kode Skenario NS-2.

Konfigurasi untuk *file traffic* bisa dilakukan dengan membuat *file* berekstensi .txt untuk menyimpan konfigurasi tersebut. Pada *file* konfigurasi lingkungan simulasi, *file traffic* tersebut dimasukkan agar dibaca sebagai *file traffic*. Potongan konfigurasi *file traffic* dapat dilihat pada Gambar 4.16.

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(58) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(59) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
```

Gambar 4.16 Implementasi Simulasi File Traffic

Pada konfigurasi tersebut, ditentukan *node* sumber dan *node* tujuan pengiriman paket. Pengiriman dimulai pada detik ke- 2.55. Implementasi konfigurasi *file traffic* untuk simulasi pada NS-2 dapat dilihat pada lampiran A.5 Kode Konfigurasi *Traffic*

4.4 Implementasi Metrik Analisis

Simulasi yang telah dijalankan oleh NS-2 menghasilkan sebuah *trace file* yang berisikan data mengenai apa saja yang terjadi selama simulasi dalam bentuk *file* berekstensi .tr. Dari data *trace file* tersebut, dapat dilakukan analisis performa *routing protocol* dengan mengukur beberapa metrik. Pada Tugas Akhir ini, metrik yang akan dianalisis adalah PDR, E2E, dan RO, dan Forwarded Route Request (RREQ F).

4.4.1 Implementasi *Packet Delivery Ratio* (PDR)

Pada subbab 2.3.2 telah ditunjukkan contoh struktur data *event* yang dicatat dalam *trace file* oleh NS-2. Kemudian, pada persamaan 3.1 telah dijelaskan bagaimana menghitung PDR. Skrip awk untuk menghitung PDR berdasarkan kedua informasi tersebut

dapat dilihat pada lampiran A.6 Kode Skrip AWK *Packet Delivery Ratio*.

PDR didapatkan dengan cara menghitung setiap baris terjadinya *event* pengiriman dan penerimaan paket data yang dikirim melalui agen pada *trace file*. Skrip menyaring setiap baris yang mengandung *string* AGT karena kata kunci tersebut menunjukan *event* yang berhubungan dengan paket komunikasi data. Penghitungan dilakukan dengan menjumlahkan paket yang dikirimkan dan paket yang diterima dengan menggunakan karakter pada kolom pertama sebagai *filter*. Kolom pertama menunjukan event yang terjadi dari sebuah paket. Setelah itu nilai PDR dihitung dengan cara persamaan 3.1. Pseudocode untuk menghitung PDR dapat dilihat pada Gambar 4.17.

```
sent = 0
received = 0
for i = 1 to the number of rows
     if in a row contains "s" and AGT then
          sent++
     else if in a row contains "r" and AGT then
          received++
     end if
pdr = received / sent
```

Gambar 4.17 Pseudocode untuk Menghitung PDR

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f pdr.awk tracefile.tr`.

4.4.2 Implementasi *Average End-to-End Delay (E2E)*

Skrip awk untuk menghitung E2E dapat dilihat pada lampiran A.7 Kode Skrip AWK Rata-Rata *End-to-End Delay*.

Dalam perhitungan E2E, langkah yang digunakan untuk mendapatkan E2E hampir sama dengan ketika mencari PDR, hanya saja yang perlu diperhatikan adalah waktu dari sebuah *event* yang tercatat pada kolom ke-2 dengan *filter event* pada kolom ke-4 adalah layer AGT dan *event* pada kolom pertama guna membedakan paket dikirim atau diterima. Setelah seluruh baris yang memenuhi didapatkan, akan dihitung *delay* dari paket dengan mengurangi waktu dari paket diterima dengan waktu dari paket dikirim dengan syarat memiliki *id* paket yang sama.

Setelah mendapatkan *delay* paket, langkah selanjutnya adalah dengan mencari rata-rata dari *delay* tersebut dengan menjumlahkan semua *delay* paket dan membaginya dengan jumlah paket. *Pseudocode* untuk menghitung rata-rata E2E dapat dilihat pada Gambar 4.18.

```
sum_delay = 0
counter = 0
for i = 1 to the number of rows
     counter++
     if layer == AGT and event == s then
      start_time[packet_id] = time
     else if layer == AGT and event == r then
      end_time[packet_id] = time
     end if
     delay[packet_id] = end_time[packet_id] –
start_time[packet_id]
     sum_delay += delay[packet_id]
e2e = sum_delay / counter
```

Gambar 4.18 Pseudocode untuk Perhitungan Rata-Rata E2E

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f e2e.awk tracefile.tr`.

4.4.3 Implementasi *Routing Overhead* (RO)

Seperti yang telah dijelaskan sebelumnya, *routing overhead* merupakan jumlah dari paket kontrol *routing* baik itu RREQ, RREP, maupun RERR. Dengan begitu, untuk mendapatkan RO yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama dan *event layer* RTR pada kolom ke-4. Perhitungan RO telah dijelaskan pada persamaan 3.3. Skrip AWK untuk menghitung RO dapat dilihat pada lampiran A.8 Kode Skrip AWK *Routing Overhead*. *Pseudocode* untuk menghitung RO dapat dilihat pada Gambar 4.19.

```
ro = 0
for i = 1 to the number of rows
     if in a row contains "s" and RTR then
        ro++
     end if
```

Gambar 4.19 Pseudocode untuk Perhitungan *Routing Overhead*

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f ro.awk scenario1.tr`.

4.4.4 Implementasi *Forwarded Route Request* (RREQ F)

Forwarded Route Request (RREQ F) adalah jumlah paket *control route request* yang diteruskan per-data paket ke *node* tujuan selama simulasi terjadi. Dengan begitu, untuk mendapatkan RREQ F yang perlu dilakukan adalah menjumlahkan tiap paket dengan *filter event sent* pada kolom pertama, *event layer* RTR pada kolom ke-4, dan *event layer route request* pada kolom-25. Penyaringan juga dilakukan pada kolom ke-3 yang menunjukan *node* id. Selama *node*

bukan *node* sumber, maka akan terus dilakukan penjumlahan baris yang terdapat pada *file* dengan ekstensi .tr. Perhitungan RREQ F telah dijelaskan pada persamaan 3.4. Skrip awk untuk menghitung RREQ F dapat dilihat pada lampiran A.9 Kode Skrip AWK *Forwarded Route Request* . Pseudocode untuk menghitung RREQ F dapat dilihat pada Gambar 4.20.

```
rreqf = 0
for i = 1 to the number of rows
     if packet is AODV and RREQ and not source
node then
        rreqf++
     end if
```

Gambar 4.20 Pseudocode Perhitungan *Forwarded Route Request*

Contoh perintah pengeksekusian skrip awk untuk menganalisis *trace file* adalah `awk -f rreqf.awk scenario1.tr.`

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dilakukan tahap ujicoba dan evaluasi sesuai dengan rancangan dan implementasi. Dari hasil yang didapatkan setelah melakukan uji coba, akan dilakukan evaluasi sehingga dapat ditarik kesimpulan pada bab selanjutnya.

5.1 Lingkungan Uji Coba

Uji coba dilakukan pada perangkat dengan spesifikasi seperti yang tertera pada Tabel 5.1.

Tabel 5.1 Spesifikasi Perangkat yang Digunakan

Komponen	**Spesifikasi**
CPU	Intel(R) Core (TM) i3-2120 CPU @ 3.30GHz x 4
Sistem Operasi	Ubuntu 16.04 LTS
Linux Kernel	Linux kernel 4.4
Memori	8.0 GB

Adapun versi perangkat lunak yang digunakan dalam Tugas Akhir ini adalah sebagai berikut:

- SUMO versi 0.25.0 untuk pembuatan skenario mobilitas VANETs.
- JOSM versi 10301 untuk penyuntingan peta OpenStreetMap.
- NS-2 versi 2.35 untuk simulasi skenario VANETs.

Parameter lingkungan uji coba yang digunakan pada NS-2 dapat diihat pada Tabel 5.2. Pengujian dilakukan dengan menjalankan skenario yang disimulasikan pada NS-2. Dari simulasi tersebut dihasilkan sebuah *trace file* dengan ekstensi .tr yang akan dianalisis dengan bantuan skrip awk untuk mendapatkan PDR, E2E, RO, dan RREQ F menggunakan kode yang terdapat pada lampiran A.6 Kode Skrip AWK *Packet Delivery Ratio*, A.7 Kode Skrip AWK Rata-Rata

End-to-End Delay, A.8 Kode Skrip AWK *Routing Overhead*, dan A.9 Kode Skrip AWK *Forwarded Route Request*.

Tabel 5.2 Lingkungan Uji Coba

No.	Parameter	Spesifikasi
1	Network simulator	NS-2.35
2	Routing protocol	AODV
3	Waktu simulasi	200 detik
4	Area simulasi	1500 m x 1500 m
5	Jumlah *Node*	60, 150, 300
6	Radius transmisi	400m
7	Kecepatan maksimum	20 m/s
8	Protokol MAC	IEEE 802.11p
9	Model Propagasi	*Two-ray ground*

5.2 Hasil Uji Coba

Hasil pra uji coba penentuan *threshold* , uji coba skenario *grid*, dan skenario *real* untuk Tugas Akhir ini dapat dilihat sebagai berikut:

5.2.1 Hasil Pra Uji Coba Penentuan *Threshold*

Sebelum melakukan uji coba, dilakukan pra uji coba untuk menentukan nilai *threshold* yang akan dijadikan sebagai perbandingan dengan node tetangga. Penentuan nilai *threshold* akan dilakukan dengan cara melakukan simulasi dengan *threshold* yang diambil dari angka satu hingga angka yang membuat nilai *Packet Delivery Ratio* (PDR), *Routing Overhead* (RO), dan *Forwarded Route Request* (RREQ F) pada AODV modifikasi turun secara drastis dibandingkan dengan AODV asli. Pra uji coba dilakukan selama 10 kali dan dilihat hasil metrik analisis yang terbaik dan muncul paling banyak pada 10 skenario tersebut. Hasil yang unggul tersebut akan dijadikan nilai *threshold* untuk diimplementasikan pada lingkungan uji coba. Dari hasil pra uji coba, didapatkan nilai *threshold* 9 untuk lingkungan 60 *node* (jarang), nilai *threshold* 21 untuk lingkungan 150

node (sedang), dan nilai *threshold* 45 untuk lingkungan 300 *node* (padat). Berikut contoh hasil dari pra uji coba untuk lingkungan jarang, sedang, dan padat yang dapat dilihat pada Tabel 5.3, Tabel 5.4, dan Tabel 5.5.

Tabel 5.3 Hasil Pra Uji Coba di Lingkungan Jarang

Threshold	***Packet Delivery Ratio***	***Routing Overhead***	***Forwarded Route Request***	***End-to-End Delay***
0	0,7795	14822	2173	1007,36 ms
5	0,7761	14289	1893	797,493 ms
6	0,6378	14467	2053	830,054 ms
7	0,7846	13955	1536	719,429 ms
8	0,7835	13452	1193	780,241 ms
9	0,8159	14457	2083	1603,7 ms

Tabel 5.4 Hasil Pra Uji Coba di Lingkungan Sedang

Threshold	***Packet Delivery Ratio***	***Routing Overhead***	***Forwarded Route Request***	***End-to-End Delay***
0	0,7795	14822	2173	1007,36 ms
5	0,7761	14289	1893	797,493 ms
6	0,6378	14467	2053	830,054 ms
7	0,7846	13955	1536	719,429 ms
8	0,7835	13452	1193	780,241 ms
9	0,8159	14457	2083	1603,7 ms

Tabel 5.5 Hasil Pra Uji Coba di Lingkungan Padat

Threshold	***Packet Delivery Ratio***	***Routing Overhead***	***Forwarded Route Request***	***End-to-End Delay***
0	0,7795	14822	2173	1007,36 ms
5	0,7761	14289	1893	797,493 ms
6	0,6378	14467	2053	830,054 ms
7	0,7846	13955	1536	719,429 ms

Threshold	*Packet Delivery Ratio*	*Routing Overhead*	*Forwarded Route Request*	*End-to-End Delay*
8	0,7835	13452	1193	780,241 ms
9	0,8159	14457	2083	1603,7 ms

5.2.2 Hasil Uji Coba Skenario *Grid*

Pengujian pada skenario *grid* digunakan untuk melihat perbandingan PDR, RO, RREQ F, dan E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, RO, RREQ F, dan E2E pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas *random* pada peta *grid* dengan luas area 1300 m x 1300 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 *node* untuk lingkungan yang sedang, dan 300 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Untuk uji coba setiap lingkungan menggunakan *threshold* yang berbeda-beda karena satu *threshold* tidak bisa disamakan untuk semua lingkungan disebabkan jumlah tetangga yang dimiliki oleh setiap *node* pada setiap lingkungan berbeda-beda, tergantung pada jumlah *node* pada lingkungan simulasi tersebut. Pencarian nilai *threshold* dimulai dari angka 1 hingga angka yang membuat nilai PDR, RO, dan RREQ F mengalami penurunan yang drastis.

Untuk lingkungan jarang dengan jumlah *node* 60, akan digunakan *threshold* 9. Untuk lingkungan sedang dengan jumlah *node* 150, akan dibandingkan kinerja AODV asli dengan AODV yang dimodifikasi menggunakan *threshold* 21. Sedangkan untuk lingkungan padat dengan jumlah *node* 300 akan dibandingkan AODV asli dengan AODV yang dimodifikasi dengan *threshold* 45. Hasil analisis dapat dilihat pada Tabel 5.6, Tabel 5.7, Tabel 5.8, dan Tabel 5.9.

Tabel 5.6 Hasil Rata - Rata PDR Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	0,7879	0,7659	0,022
150	0,8473	0,8831	0,0358
300	0,8519	0,9403	0,0884

Tabel 5.7 Hasil Rata - Rata E2E Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	865,3	1138,4	273,1
150	620,1	378,7	241,4
300	739,56	274,4	465,1

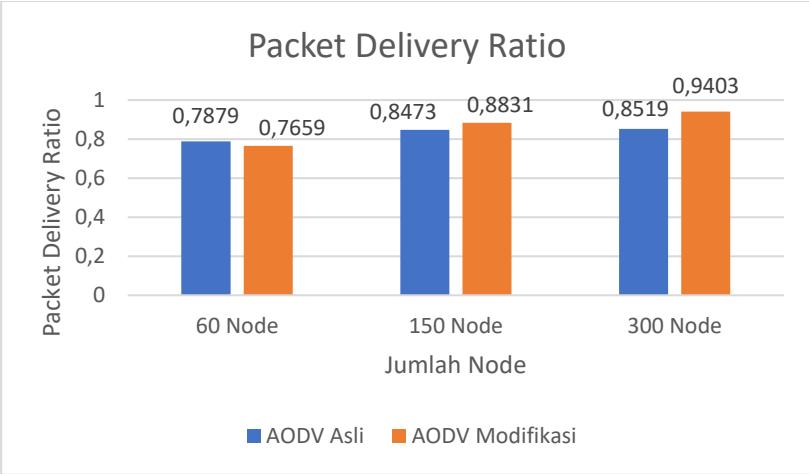
Tabel 5.8 Hasil Rata - Rata RO Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	14594,8	14178,4	416,4
150	41972,4	39943	2029,4
300	66379,6	62080,2	4299,4

Tabel 5.9 Hasil Rata - Rata RREQ F Skenario Grid

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	2010	1727	283
150	4601,2	2591,2	2010
300	6408,4	4138	2270,4

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, RO, RREQ F, dan E2E yang ditunjukkan pada Gambar 5.1, Gambar 5.2, Gambar 5.3, dan Gambar 5.4

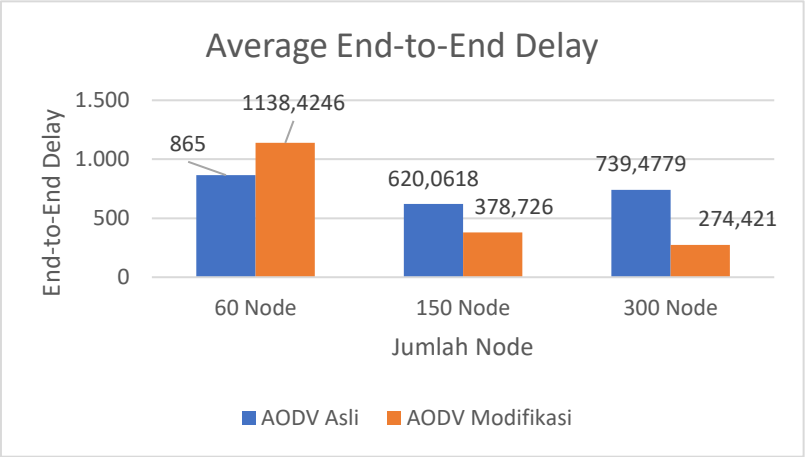


Gambar 5.1 Grafik PDR Skenario *Grid*

Berdasarkan grafik pada Gambar 5.1, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan yang tidak terlalu signifikan pada PDR. Pada lingkungan yang jarang dengan *node* berjumlah 60, menghasilkan perbedaan selisih PDR sebesar 0,022, dimana terjadi penurunan sebesar 2,8% dan *routing protocol* AODV yang asli unggul dalam hal PDR tersebut. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0,0358, dimana terjadi kenaikan PDR sebesar 4,22% dan *routing protocol* AODV yang dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih PDR sebesar 0,0884, dimana terjadi kenaikan PDR sebesar 10,37% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang banyak atau pada lingkungan yang padat, menghasilkan PDR yang lebih baik daripada di lingkungan dengan jumlah *node* yang jarang maupun sedang baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah

dimodifikasi. Nilai rata-rata kenaikan PDR pada skenario *grid* adalah sebesar 3,93%.

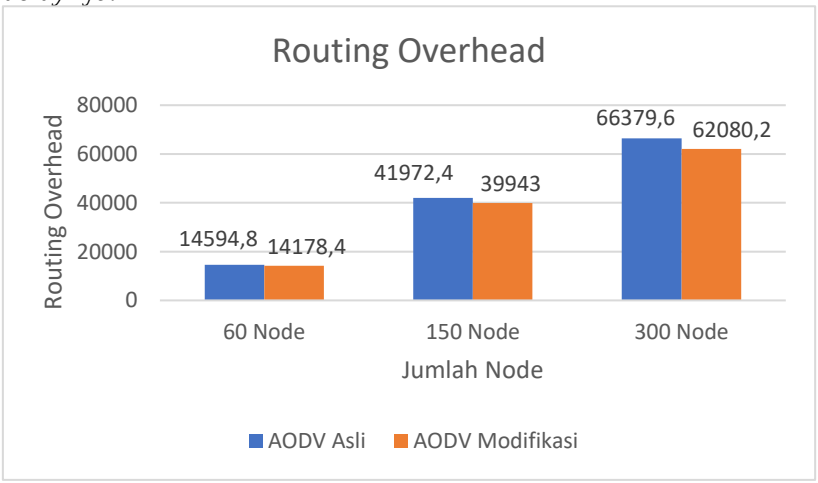


Gambar 5.2 Grafik E2E Skenario *Grid*

Berdasarkan grafik pada Gambar 5.2, dapat dilihat bahwa rata-rata E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 60 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 273,1 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi, dimana *routing protocol* AODV asli unggul dalam hal E2E tersebut. Sedangkan pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 241,4 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi, dimana *routing protocol* AODV yang telah dimodifikasi lebih unggul dalam hal E2E. Pada lingkungan yang padat dengan jumlah 300 *node*, terjadi perbedaan selisih E2E sebesar 465,1 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi, dimana *routing protocol* AODV

yang telah dimodifikasi jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Jika ketiga lingkungan tersebut dibandingkan, memang pada lingkungan yang jarang, sedang, maupun padat tidak selalu lebih unggul dalam hal E2E. Routing protocol AODV yang telah dimodifikasi tidak selalu lebih unggul daripada *routing protocol* AODV yang telah dimodifikasi. Hasil rata – rata E2E tidak dapat dianalisis karena terjadi fluktuasi dan tidak stabil. Hal ini dikarenakan waktu *delay* tergantung dari rata – rata waktu paket yang terkirim. Semakin banyak paket yang terkirim, maka semakin beragam *delay*nya.



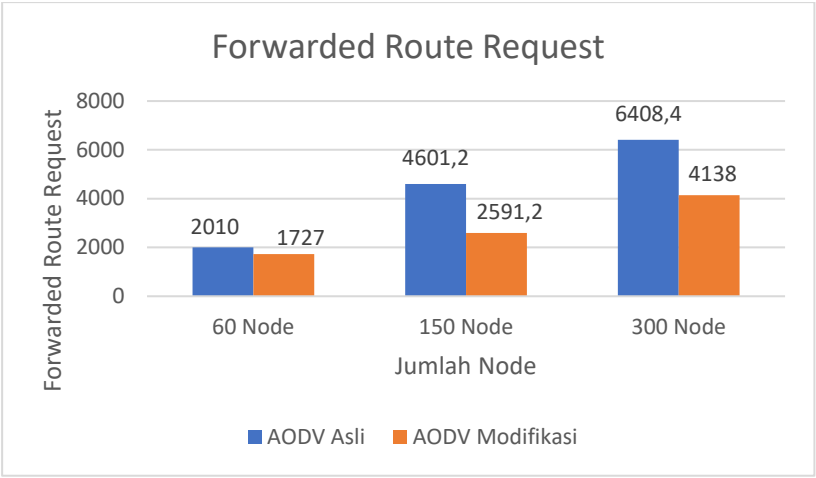
Gambar 5.3 Grafik Routing Overhead Skenario *Grid*

Berdasarkan grafik pada Gambar 5.3, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* AODV asli mengalami kenaikan yang signifikan pada *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih RO sebesar 416,4, dimana terjadi penurunan RO sebesar 2,85% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli. Pada

lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *routing overhead* sebesar 2029,4, dimana terjadi penurunan RO sebesar 4,8% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal RO tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih RO sebesar 4299,4, dimana terjadi penurunan RO sebesar 6,47% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal RO tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan RO yang lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Nilai rata – rata penurunan RO pada AODV yang dimodifikasi adalah sebesar 4,7%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan RO yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli.

Untuk hasil pengambilan data *forwarded route request* (RREQ F) pada skenario *grid* 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.4.



Gambar 5.4 Grafik *Forwarded Route Request* Skenario *Grid*

Berdasarkan grafik pada Gambar 5.4, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang signifikan. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 283, dimana terjadi penurunan RREQ F sebesar 14,07% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 2010, dimana terjadi penurunan RREQ F sebesar 43,68% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam RREQ F tersebut dari RREQ F AODV asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih RREQ F sebesar 2270.4, dimana terjadi penurunan RREQ F sebesar 35,43% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal RREQ F tersebut.

Jika ketiga lingkungan tersebut dibandingkan, dapat dilihat bahwa dengan jumlah *node* yang sedikit atau lingkungan jarang

menghasilkan RREQ F yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Nilai rata – rata penurunan yang terjadi pada RREQ F adalah sebesar 31,06%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan RREQ F yang lebih bagus atau dalam hal ini lebih rendah daripada RREQ F asli dengan jumlah selisih RREQ F yang cukup signifikan.

5.2.3 Hasil Uji Coba Skenario *Real*

Pengujian pada skenario *real* digunakan untuk melihat perbandingan PDR, E2E, RO, dan RREQ F antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi dalam pemilihan *node* yang dapat menerima paket *route request*.

Pengambilan data uji PDR, E2E, RO, dan RREQ F pada skenario *grid* dilakukan sebanyak 10 kali dengan skenario mobilitas random pada peta *grid* dengan luas area 1500 m x 1300 m dan *node* sebanyak 60 untuk lingkungan yang jarang, 150 *node* untuk lingkungan yang sedang, dan 300 *node* untuk lingkungan yang padat dilakukan pada kecepatan standar yaitu 20 m/s. Untuk uji coba setiap lingkungan menggunakan *threshold* yang berbeda-beda karena satu *threshold* tidak bisa disamakan untuk semua lingkungan disebabkan jumlah tetangga yang dimiliki oleh setiap *node* pada setiap lingkungan berbeda-beda, tergantung pada jumlah *node* pada lingkungan simulasi tersebut. Pencarian nilai *threshold* dimulai dari angka 1 hingga angka yang membuat nilai PDR, RO, dan RREQ F mengalami penurunan yang drastis.

Untuk lingkungan tidak padat dengan jumlah *node* 60, akan digunakan *threshold* 9. Untuk lingkungan sedang dengan jumlah *node* 150, akan dibandingkan kinerja AODV asli dengan AODV yang dimodifikasi menggunakan *threshold* 21. Sedangkan untuk lingkungan padat dengan jumlah *node* 300 akan dibandingkan AODV asli dengan AODV yang dimodifikasi dengan *threshold* 45. Hasil

analisis dapat dilihat pada Tabel 5.10, Tabel 5.11, Tabel 5.12, dan Tabel 5.13.

Tabel 5.10 Hasil Rata - Rata Perhitungan PDR pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	0.8303	0.87544	0.04514
150	0.81998	0.84482	0.02484
300	0.67916	0.80904	0.12988

Tabel 5.11 Hasil Rata -Rata Perhitungan E2E pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	428.1874	314.9866	113.2008
150	608.06924	84.24624	523.823
300	366.31188	3137.6682	2771.35632

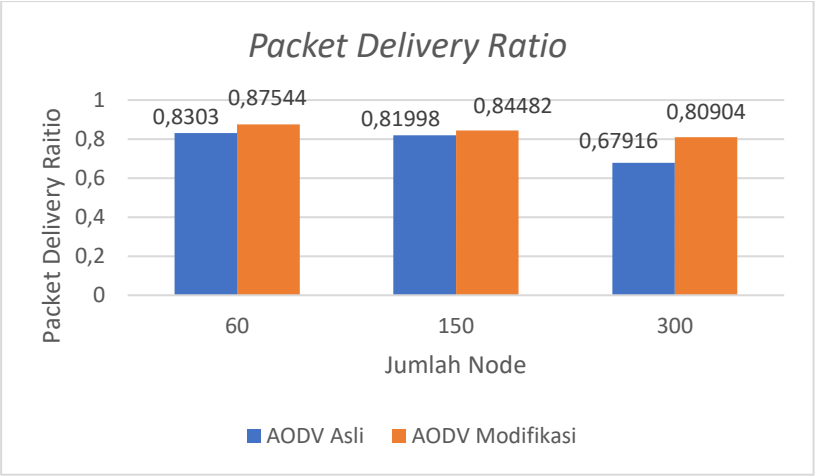
Tabel 5.12 Hasil Rata - Rata Perhitungan RO pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	14578.4	14223.2	2144
150	39690.6	38943.4	747.2
300	62547.6	61164.8	1382.8

Tabel 5.13 Hasil Rata - Rata Perhitungan RREQ F pada Skenario Real

Jumlah *Node*	**AODV Asli**	**AODV Modifikasi**	**Perbedaan**
60	2144	1839.6	304.4
150	2187	1504	683
300	2772.4	1401.8	1370.6

Dari data di atas, dibuat grafik yang merepresentasikan hasil perhitungan PDR, E2E, RO, dan RREQ F yang ditunjukkan pada Gambar 5.5, Gambar 5.6, Gambar 5.7, dan Gambar 5.8

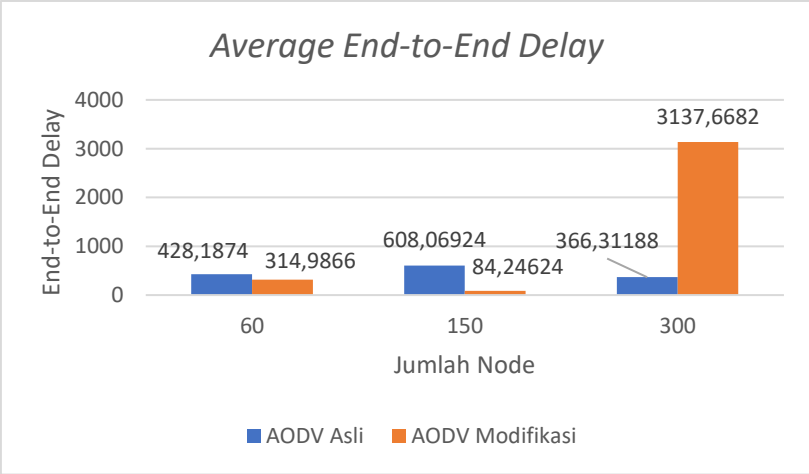


Gambar 5.5 Grafik Rata - Rata PDR Skenario *Real*

Berdasarkan grafik pada Gambar 5.5, dapat dilihat bahwa *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami penurunan yang signifikan pada PDR. Pada lingkungan yang jarang dengan *node* berjumlah 60, menghasilkan perbedaan selisih PDR sebesar 0,04514, dimana terjadi kenaikan PDR sebesar 5,43% dan *routing protocol* AODV yang dimodifikasi unggul dalam hal PDR tersebut. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih PDR sebesar 0,02484, dimana terjadi kenaikan PDR sebesar 3,02% dan *routing protocol* AODV yang dimodifikasi unggul dalam hal PDR tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih PDR sebesar 0,12988, dimana terjadi kenaikan sebesar 19,12% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal PDR tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa pada lingkungan yang jarang, menghasilkan PDR yang lebih bagus daripada di lingkungan dengan jumlah *node* yang sedang maupun padat baik untuk *routing protocol* AODV asli maupun

routing protocol AODV yang telah dimodifikasi. Nilai rata-rata kenaikan PDR pada skenario *real* adalah sebesar 9,19%.

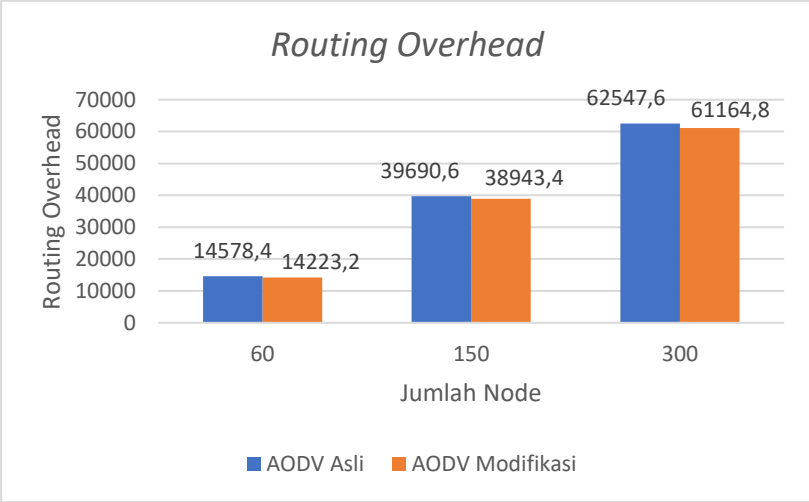


Gambar 5.6 Grafik E2E pada Skenario *Real*

Berdasarkan grafik pada Gambar 5.6, dapat dilihat bahwa rata-rata E2E antara *routing protocol* AODV asli dan AODV yang telah dimodifikasi mengalami perubahan yang fluktuatif. Pada lingkungan yang jarang dengan jumlah 60 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 113.2008 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi, dimana *routing protocol* AODV yang dimodifikasi unggul dalam hal E2E tersebut. Sedangkan pada lingkungan sedang dengan jumlah 150 *node*, terjadi perbedaan selisih *end-to-end delay* sebesar 523.823 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi, dimana *routing protocol* AODV yang tleah dimodifikasi lebih unggul dalam hal E2E. Pada lingkungan yang padat dengan jumlah 300 *node*, terjadi perbedaan selisih E2E sebesar 2771.35632 ms antara *routing protocol* AODV asli dengan *routing protocol* AODV yang telah dimodifikasi, dimana *routing protocol*

AODV yang asli jauh lebih unggul dalam hal *end-to-end delay* tersebut.

Jika ketiga lingkungan tersebut dibandingkan, memang pada lingkungan yang jarang, sedang, maupun padat tidak selalu lebih unggul dalam hal E2E. Hasil rata – rata E2E tidak dapat dianalisis karena terjadi fluktuasi dan tidak stabil. Hal ini dikarenakan waktu *delay* tergantung dari rata – rata waktu paket yang terkirim. Semakin banyak paket yang terkirim, maka semakin beragam *delay*nya.



Gambar 5.7 Grafik Rata - Rata RO Skenario *Real*

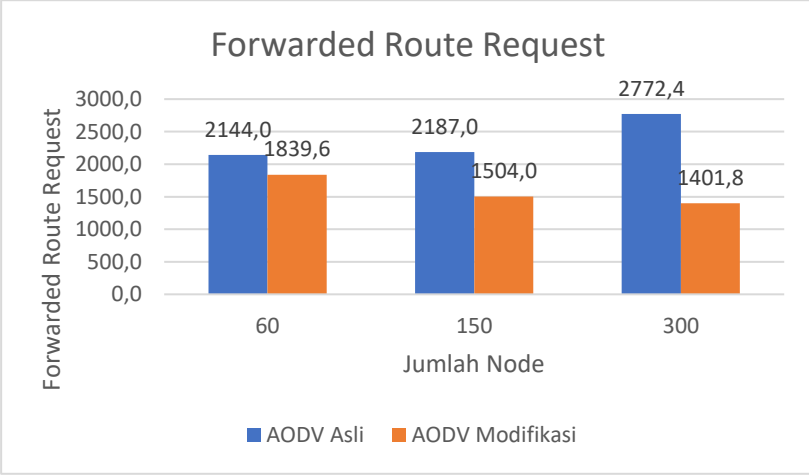
Berdasarkan grafik pada

Gambar 5.7, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* AODV asli mengalami kenaikan *routing overhead*. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih RO sebesar 2144, dimana terjadi penurunan RO sebesar 2,43% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam hal RO tersebut karena menghasilkan RO yang lebih rendah dari AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan

perbedaan selisih *routing overhead* sebesar 747.2, dimana terjadi penurunan RO sebesar 1,88% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam hal RO tersebut dari AODV asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih RO sebesar 1382.8, dimana terjadi penurunan RO sebesar 2,21% dan routing protocol AODV yang telah dimodifikasi juga unggul dalam hal RO tersebut.

Jika ketiga lingkungan tersebut dibandingkan dapat dilihat bahwa dengan jumlah *node* yang lebih sedikit atau pada lingkungan dengan *node* yang jarang, menghasilkan RO yang lebih sedikit daripada di lingkungan dengan jumlah *node* yang sedang maupun yang padat baik untuk *routing protocol* AODV asli maupun *routing protocol* AODV yang telah dimodifikasi. Nilai rata – rata penurunan PDR pada simulasi *real* adalah sebesar 2,17%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan RO yang lebih bagus atau dalam hal ini lebih rendah daripada AODV asli.

Untuk hasil pengambilan data *forwarded route request* (RREQ F) pada skenario *grid* 60 *node*, 150 *node*, dan 300 *node* dapat dilihat pada Gambar 5.8



Gambar 5.8 Grafik Rata - Rata RREQ F Skenario *Real*

Berdasarkan grafik pada Gambar 5.8, dapat dilihat bahwa *routing protocol* AODV yang telah dimodifikasi dan juga *routing protocol* asli mengalami perubahan *forwarded route request* (RREQ F) yang signifikan. Pada lingkungan yang jarang dengan jumlah 60 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 304.4, dimana terjadi penurunan RREQ F sebesar 14,20% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam hal RREQ F tersebut karena menghasilkan RREQ F yang lebih rendah dari *routing protocol* AODV asli. Pada lingkungan yang sedang dengan jumlah 150 *node*, menghasilkan perbedaan selisih *forwarded route request* sebesar 683, dimana terjadi penurunan RREQ F sebesar 31,23% dan *routing protocol* AODV yang telah dimodifikasi unggul dalam RREQ F tersebut dari RREQ F AODV asli. Pada lingkungan yang padat dengan jumlah 300 *node*, menghasilkan perbedaan selisih RREQ F sebesar 1370.6, dimana terjadi penurunan RREQ F sebesar 49,44% dan *routing protocol* AODV yang telah dimodifikasi juga unggul dalam hal RREQ F tersebut.

Jika ketiga lingkungan tersebut dibandingkan, dapat dilihat bahwa pada lingkungan sedang menghasilkan RREQ F yang lebih bagus atau lebih sedikit daripada di lingkungan dengan jumlah *node* yang jarang maupun yang padat untuk *routing protocol* AODV asli . Sedangkan untuk *routing protocol* AODV yang telah dimodifikasi, lingkungan padat menghasilkan RREQ F yang lebih baik daripada lingkungan jarang maupun sedang. Nilai rata-rata penurunan RREQ F pada skenario *real* adalah sebesar 31,62%. Dapat dilihat pula bahwa AODV yang telah dimodifikasi menghasilkan RREQ F yang lebih bagus atau dalam hal ini lebih rendah daripada RREQ F asli dengan jumlah selisih RREQ F yang cukup signifikan.

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Pada Bab ini akan diberikan kesimpulan yang diperoleh dari Tugas Akhir yang telah dikerjakan dan saran tentang pengembangan dari Tugas Akhir ini yang dapat dilakukan di masa yang akan datang.

6.1 Kesimpulan

Kesimpulan yang diperoleh pada uji coba dan evaluasi Tugas Akhir ini adalah sebagai berikut:

1. AODV yang dimodifikasi sudah berhasil membatasi jumlah *forwarding node* yang bertugas untuk *rebroadcast* paket RREQ.
2. Dampak pembatasan *forwarding node* terhadap performa protokol AODV secara keseluruhan adalah rata – rata kenaikan *Packet Delivery Ratio* (PDR) sebesar 9,19%, rata – rata penurunan *Routing Overhead* (RO) sebesar 2,17%, dan rata – rata penurunan *Forwarded Route Request* (RREQ F) sebesar 31,62%.

6.2 Saran

Saran yang dapat diberikan dari hasil uji coba dan evaluasi adalah sebagai berikut:

1. Lebih banyak uji coba yang dilakukan untuk mendapatkan hasil yang lebih akurat.
2. Perhitungan jumlah *node* tetangga yang lebih dinamis pada *routing protocol* AODV. Contohnya saat *node* tetangga menjauh dari jangkauan, maka ada mekanisme untuk memperbarui *list* simpanan *node* tetangga sehingga perhitungan jumlah *node* tetangga lebih akurat.
3. Menambahkan aspek lain untuk melakukan pembatasan *forwarding node* yang meneruskan paket RREQ seperti arah, kecepatan, dan energi.

(Halaman ini sengaja dikosongkan)

DAFTAR PUSTAKA

[1] "VANET - Vehicle Ad hoc Network," [Online]. Available: http://comp.ist.utl.pt/~rmr/WSN/CaseStudies2007-no/WSN_Transportation/. [Diakses 15 November 2017].

[2] J. Harri, F. Filali dan C. Bonnet, "Mobility Models for Vehicular Ad Hoc Network: A Survey and Taxonomy," IEEE, Florida, 2009.

[3] A. Rhim dan Z. Dziong, "Routing Based on Link Expiration Time for MANET Performance Improvement," IEEE, Kuala Lumpur, 2009.

[4] P. Jacquet, P. Muhlethaler, T. Clausen, A. Laouiti, A. Qayyum dan L. Viennot, "Optimized Link State Routing Protocol for Ad hoc Networks," IEEE, Lahore, 2001.

[5] S. N. Ferdous dan M. S. Hossain, "Randomized Energy-Based AODV Protocol for Wireless Ad-Hoc Network," IEEE, Dhaka, 2016.

[6] R. Brendha dan V. S. J. Prakash, "A Survey on Routing Protocols for Vehicular Ad hoc Networks," IEEE, Coimbatore, 2017.

[7] R. F. Sari dan A. Syarif, "Analisis Kinerja Protokol Routing Ad Hoc On-Demand Distance Vector (AODV) pada Jaringan Ad Hoc," p. 22, October 2010.

[8] P. Meeneghan dan D. Delaney, "An Introduction to NS Nam and OTcl scripting," April 2004.

[9] "OpenStreetMap," [Online]. Available: https://www.openstreetmap.org/. [Diakses 15 November 2017].

[10] "JOSM," [Online]. Available: https://josm.openstreetmap.de/. [Diakses 15 November 2017].

[11] D. Krajzewics, J. Erdmann, M. Behrisch dan L. Bieker, "Recent Development and Application of SUMO,"

International Journal On Advances in Systems and Measurements, p. 128, December 2012.

[12] "AWK," [Online]. Available: http://tldp.org/LDP/abs/html/awk.html. [Diakses 10 01 2017].

[13] R. G. Engoulou, M. Bellaiche, S. Pierre dan A. Quintero, "VANET Security Surveys," *Computer Communication*, vol. 44, p. 2, 2014.

[14] M. Iqbal, M. Shafiq, H. Attaullah, J.-G. Choi, K. Akram dan X. Wang, "Design and Analysis of a Novel Hybrid Wireless Mesh Network Routing Protocol," p. 22, January 2014.

LAMPIRAN

A.1 Kode Fungsi addNeighbor()

```
void Node::addNeighbor(Node * neighbor) {
      int flag=0;
      neighbor_list_node*my_neighbor_list;
      my_neighbor_list=neighbor_list_;

      while(my_neighbor_list){
            if(my_neighbor_list->nodeid ==
            neighbor->nodeid())
            {
            flag = 1;
            break;
            }
            else{
                  my_neighbor_list=my_neigh
            bor_list->next;
            }
      }

      if(flag == 0){
            neighbor_list_node* nlistItem =
            (neighbor_list_node
            *)malloc(sizeof(neighbor_list_n
            ode));
            nlistItem->nodeid = neighbor-
            >nodeid();
            nlistItem->next =
            neighbor_list_;
            neighbor_list_=nlistItem;
      }
}
```

A.2 Kode Fungsi recvHello()

```
void AODV::recvHello(Packet *p)
{
  struct hdr_aodv_reply *rp =
  HDR_AODV_REPLY(p);
  AODV_Neighbor *nb;

  Node *sender_node =
  Node::get_node_by_address(rp->rp_dst);
  Node *receiver_node =
  Node::get_node_by_address(index);

  nb = nb_lookup(rp->rp_dst);
  if (nb == 0)
  {
    sender_node->addNeighbor(receiver_node);
    receiver_node->addNeighbor(sender_node);

    nb_insert(rp->rp_dst);

    Node* m_node =
    Node::get_node_by_address(this->addr());
    neighbor_list_node*my_mobile_neighbor_lis
    t;
    my_mobile_neighbor_list = m_node-
    >neighbor_list_;
    int count = 0;
   }
   else{
    nb->nb_expire = CURRENT_TIME + (1.5 *
    ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
  }
  Packet::free(p);
}
```

A.3 Kode Fungsi recvRequest()

```
void AODV::recvRequest(Packet *p)
{
  struct hdr_ip *ih = HDR_IP(p);
  struct hdr_aodv_request *rq
  HDR_AODV_REQUEST(p);
  aodv_rt_entry *rt;

  Node* m_node =
  Node::get_node_by_address(this->addr());
  neighbor_list_node*
  my_mobile_neighbor_list;
  my_mobile_neighbor_list = m_node-
  >neighbor_list_;
  int count = 0;

  while(my_mobile_neighbor_list)
  {
     count++;
     if(my_mobile_neighbor_list->next){
      my_mobile_neighbor_list=my_mobile_neigh
      bor_list->next;
     }
     else{
        break;
     }
  }

  if (rq->rq_src == index)
  {
      #ifdef DEBUG
          fprintf(stderr, "%s: got my own
      REQUEST\n", __FUNCTION__);
      #endif // DEBUG
      Packet::free(p);
      return;
  }
```

```
  if (id_lookup(rq->rq_src, rq-
>rq_bcast_id)){
       #ifdef DEBUG
       fprintf(stderr, "%s: discarding
request\n", __FUNCTION__);
       #endif // DEBUG

       Packet::free(p);
       return;
  }

  if (count < 45 && rq->rq_dst != index){
     Packet::free(p);
     return;
 }

 id_insert(rq->rq_src, rq->rq_bcast_id);

 aodv_rt_entry *rt0; // rt0 is the reverse
route

 rt0 = rtable.rt_lookup(rq->rq_src);
 if (rt0 == 0){
      rt0 = rtable.rt_add(rq->rq_src);
 }
 rt0->rt_expire = max(rt0->rt_expire,
(CURRENT_TIME +
 REV_ROUTE_LIFE));

 if ((rq->rq_src_seqno > rt0->rt_seqno) ||
 ((rq->rq_src_seqno == rt0->rt_seqno) &&
 (rq->rq_hop_count < rt0->rt_hops))){
     rt_update(rt0, rq->rq_src_seqno, rq-
>rq_hop_count, ih->saddr(),
               max(rt0->rt_expire,
(CURRENT_TIME + REV_ROUTE_LIFE)));
```

```
   if (rt0->rt_req_timeout > 0.0)
      {
        rt0->rt_req_cnt = 0;
        rt0->rt_req_timeout = 0.0;
        rt0->rt_req_last_ttl = rq-
>rq_hop_count;
        rt0->rt_expire = CURRENT_TIME +
ACTIVE_ROUTE_TIMEOUT;
      }

      assert(rt0->rt_flags == RTF_UP);
      Packet *buffered_pkt;
      while ((buffered_pkt =
rqueue.deque(rt0->rt_dst)))
      {
        if (rt0 && (rt0->rt_flags == RTF_UP))
        {
          assert(rt0->rt_hops != INFINITY2);
          forward(rt0, buffered_pkt,
NO_DELAY);
        }
      }
    }
    rt = rtable.rt_lookup(rq->rq_dst);
    if (rq->rq_dst == index)
    {

  #ifdef DEBUG
      fprintf(stderr, "%d - %s: destination
sending reply\n",
              index, _FUNCTION_);
  #endif // DEBUG

     seqno = max(seqno, rq->rq_dst_seqno) +
1;
      if (seqno % 2)
        seqno++;
```

```
      sendReply(rq->rq_src,         // IP
Destination
                1,                  // Hop
Count
                index,              // Dest IP
Address
                seqno,              // Dest
Sequence Num
                MY_ROUTE_TIMEOUT,   //
Lifetime
                rq->rq_timestamp); //
timestamp

      Packet::free(p);
    }
    else if (rt && (rt->rt_hops != INFINITY2)
&&
             (rt->rt_seqno >= rq-
>rq_dst_seqno))
    {

     assert(rq->rq_dst == rt->rt_dst);
      sendReply(rq->rq_src,
                rt->rt_hops + 1,
                rq->rq_dst,
                rt->rt_seqno,
                (u_int32_t)(rt->rt_expire -
CURRENT_TIME),
                rq->rq_timestamp);
      rt->pc_insert(rt0->rt_nexthop); //
nexthop to RREQ source
      rt0->pc_insert(rt->rt_nexthop); //
nexthop to RREQ destination

  #ifdef RREQ_GRAT_RREP
```

```
   sendReply(rq->rq_dst,
                 rq->rq_hop_count,
                 rq->rq_src,
                 rq->rq_src_seqno,
                 (u_int32_t)(rt->rt_expire -
CURRENT_TIME),
                 //             rt->rt_expire
- CURRENT_TIME,
                 rq->rq_timestamp);
  #endif
     Packet::free(p);
    else
    {
      ih->saddr() = index;
      ih->daddr() = IP_BROADCAST;
      rq->rq_hop_count += 1;
        if (rt)
        rq->rq_dst_seqno = max(rt->rt_seqno,
rq->rq_dst_seqno);
      forward((aodv_rt_entry *)0, p, DELAY);
    }

}
```

A.4 Kode Skenario NS-2

```
set val(chan)   Channel/WirelessChannel;
set val(prop)   Propagation/TwoRayGround;
set val(netif)  Phy/WirelessPhy;
set val(mac)    Mac/802_11;
set val(ifq)    Queue/DropTail/PriQueue;
set val(ll)     LL;
set val(ant)    Antenna/OmniAntenna;
set opt(x)      1300;
set opt(y)      1300;
set val(ifqlen) 1000;
set val(nn)     60;
set val(seed)   1.0;
set val(adhocRouting)   AODV;
set val(stop)   200;
set val(cp)      "cbr1.txt";
set val(sc)      "scen1modif.txt";

set ns_          [new Simulator]

# setup topography object

set topo   [new Topography]

# create trace object for ns and nam

set tracefd     [open scenario1.tr w]
set namtrace    [open scenario1.nam w]

$ns_ trace-all $tracefd
$ns_ namtrace-all-wireless $namtrace
$opt(x) $opt(y)
```

```
# Create God
set god_ [create-god $val(nn)]

#global node setting
$ns_ node-config -adhocRouting
$val(adhocRouting) \
                 -llType $val(ll) \
                 -macType $val(mac) \
                 -ifqType $val(ifq) \
                 -ifqLen $val(ifqlen) \
                 -antType $val(ant) \
                 -propType $val(prop) \
                 -phyType $val(netif) \
                 -channelType $val(chan)
\
                 -topoInstance $topo \
                 -agentTrace ON \
                 -routerTrace ON \
                 -macTrace ON \
                 -movementTrace ON \

# 802.11p default parameters
Phy/WirelessPhy set  RXThresh_ 5.57189e-
11 ; #400m
Phy/WirelessPhy set  CSThresh_ 5.57189e-
11 ; #400m

#  Create the specified number of nodes
[$val(nn)] and "attach" them
#  to the channel.
for {set i 0} {$i < $val(nn)} {incr i} {
          set node_($i) [$ns_ node]
          $node_($i) random-motion 0 ;#
disable random motion
}
```

```
# Define node movement model
puts "Loading connection pattern..."
source $val(cp)

# Define traffic model
puts "Loading scenario file..."
source $val(sc)

# Define node initial position in nam

for {set i 0} {$i < $val(nn)} {incr i} {

    # 20 defines the node size in nam,
must adjust it according to your scenario
    # The function must be called after
mobility model is defined

    $ns_ initial_node_pos $node_($i) 20
}

# Tell nodes when the simulation ends
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop).0 "$node_($i)
reset";
}

#$ns_ at  $val(stop)  "stop"
$ns_ at  $val(stop).0002 "puts \"NS
EXITING...\" ; $ns_ halt"

puts $tracefd "M 0.0 nn $val(nn) x
$opt(x) y $opt(y) rp $val(adhocRouting)"
puts $tracefd "M 0.0 sc $val(sc) cp
$val(cp) seed $val(seed)"
puts $tracefd "M 0.0 prop $val(prop) ant
$val(ant)"

puts "Starting Simulation..."
$ns_ run
```

A.5 Kode Konfigurasi *Traffic*

```
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(58) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(59) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 1
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
```

A.6 Kode Skrip AWK *Packet Delivery Ratio*

```
BEGIN {
        sendLine = 0;
        recvLine = 0;
        fowardLine = 0;
}

$0 ~/^s.* AGT/ {
        sendLine ++ ;
}

$0 ~/^r.* AGT/ {
        recvLine ++ ;
}

$0 ~/^f.* RTR/ {
        fowardLine ++ ;
}

END {
        printf "cbr s:%d r:%d, r/s
Ratio:%.4f, f:%d \n", sendLine, recvLine,
(recvLine/sendLine),fowardLine;
}
```

A.7 Kode Skrip AWK Rata-Rata *End-to-End Delay*

```
BEGIN{
      sum_delay = 0;
      count = 0;
}
{
      if ($2 >= 101) {

            if($4 == "AGT" && $1 == "s" &&
seqno < $6) {
              seqno = $6;
            }

            if($4 == "AGT" && $1 == "s") {
                  start_time[$6] = $2;
            }

            else if(($7 == "cbr") && ($1 ==
"r")) {
                  end_time[$6] = $2;
            }

            else if($1 == "D" && $7 == "cbr")
{
                  end_time[$6] = -1;
            }
      }
}
END {

    for(i=0; i<=seqno; i++) {
            if(end_time[i] > 0) {
                  delay[i] = end_time[i] -
start_time[i];
                  count++;
        }
        else {
            delay[i] = -1;
        }
    }
```

```
    for(i=0; i<=seqno; i++) {
        if(delay[i] > 0) {
           n_to_n_delay = n_to_n_delay +
delay[i];
        }
    }
     n_to_n_delay = n_to_n_delay/count;
     printf "End-to-End Delay \t= "
n_to_n_delay * 1000 " ms \n";
}
```

A.8 Kode Skrip AWK *Routing Overhead*

```
BEGIN {
     rt_pkts = 0;
}
{
           if (($1 == "s" || $1 == "f")
&& ($4 == "RTR") && ($7 == "AODV")) {

                    rt_pkts++;
           }
     }
}
END {
     printf "Routing Packets \t= %d \n",
rt_pkts;
}
```

A.9 Kode Skrip AWK *Forwarded Route Request*

```
BEGIN {
      rt_forward = 0;
}
{
            if (($1 == "s") && ($4 ==
"RTR") && ($7 == "AODV") && ($25 ==
"(REQUEST)") && ($3 != "_58_")){
                  rt_forward++;
            }
      }
}
END {
      printf "Forwarded Route Request\t=
%d \n", rt_forward;
}
```

(Halaman ini sengaja dikosongkan)

BIODATA PENULIS

Aviananda Dwirahma, lahir di Jakarta, 2 Juli 1996. Penulis adalah anak kedua dari tiga bersaudara. Penulis menempuh pendidikan sekolah dasar di SD Islamic Village Tangerang lalu melanjutkan pendidikan sekolah menengah pertama di SMP 13 Tangerang dan penulis menempuh pendidikan menengah atas di SMA Negeri 1 Tangerang. Selanjutnya penulis melanjutkan pendidikan sarjana di Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya. Selama kuliah, penulis aktif menjadi administrator Laboratorium Arsitektur dan Jaringan Komputer dan aktif dalam berbagai organisasi baik tingkat jurusan maupun fakultas. Dalam menyelesaikan pendidikan S1, penulis mengambil bidang minat Arsitektur dan Jaringan Komputer (AJK). Sebagai mahasiswa, penulis berperan aktif dalam beberapa organisasi kampus seperti staf Media Informasi Himpunan Mahasiswa Teknik-Computer (HMTC) ITS, Sekretaris Unit Kegiatan Mahasiswa Fotografi ITS (UKAFO), dan Paduan Suara Mahasiswa ITS (PSM ITS). Selain itu, penulis juga menjadi staf Desain, Dekorasi, dan Dokumentasi (3D) SCHEMATICS 2015 dan sekretaris pada acara SCHEMATICS 2016. Penulis pernah melakukan kerja praktik di PT. GMF AeroAsia periode Juni – Agustus 2017 dan membuat aplikasi berbasis web Internship Management System (IMS). Penulis dapat dihubungi melalui nomor *handphone*: 081517245353 atau *email*: avianandadj@gmail.com